eXo Platform 3.5 Reference Guides

Abstract

Welcome to eXo Platform 3.5 Reference Guide. This guide is a complete reference for customization points, and available APIs, for all modules.

Table of Contents

eXo JCR Developer Guide

Java Content Repository and Extension services

Part I. eXoJCR

Introduction in eXoJCR

JCR (JSR-170) API main concepts

Java Content Repository API as well as other Java language related standards is created within the Java Community Process http://jcp.org/ as a result of collaboration of an expert group and the Java community. It is known as *JSR-170* [http://www.jcp.org/en/jsr/detail?id=170] (Java Specification Request).

Data model

The main purpose of content repository is to maintain the data. The heart of CR is the data model:

- The main data storage abstraction of JCR's data model is a workspace
- Each repository should have one or more workspaces
- The content is stored in a workspace as a hierarchy of items
- Each workspace has its own hierarchy of items

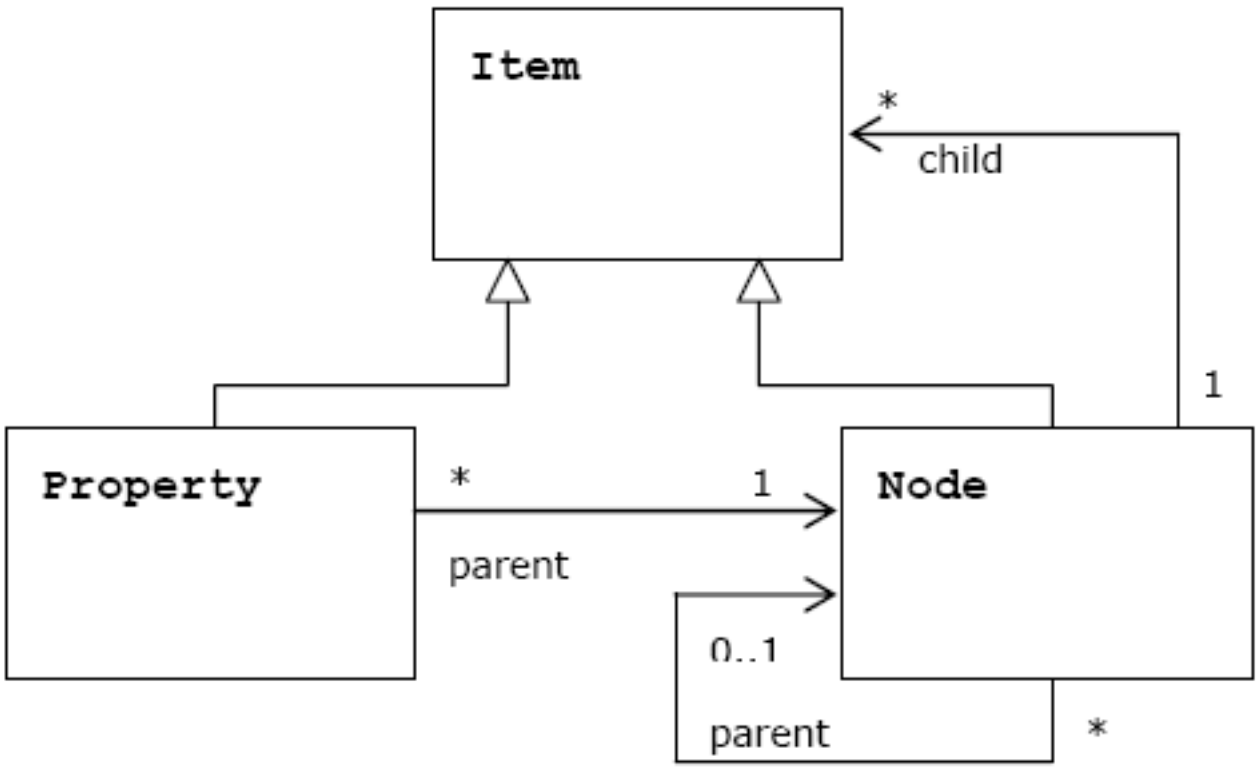


Figure 1.1. Item hierarchy

Node is intended to support the data hierarchy. It is of type using namespaced names which allows the content to be structured in accordance with standardized constraints. A node may be versioned through an associated version graph (optional feature)

Property stored data are values of predefined types (String, Binary, Long, Boolean, Double, Date, Reference, Path).

It is important to note that the data model for the interface (the repository model) is rarely the same as the data models used by the repository's underlying storage subsystems. The repository knows how to make the client's changes persistent because that is part of the repository configuration, rather than part of the application programming task.

Why use JCR?

What is JCR?

JCR (Java Content Repository) is a java interface used to access contents that are not only web contents, but also other hierarchically stored data. The content is stored in a repository. The repository can be a file system, a relational database or an XML document. The internal structure of JCR data looks similar to an XML document, that means a document tree with nodes and data, but with a small difference, in JCR the data are stored in "property items".

Or better to cite the specification of JCR: "A content repository is a high-level information management system that is a superset of traditional data repositories."

Why use JCR?

How do you know the data of your website are stored? The images are probably in a file system, the meta data are in some dedicated files - maybe in XML - the text documents and pdfs are stored in different folders with the meta data in an other place (a database?) and in a proprietary structure. How do you manage to update these data and how do you manage the access rights? If your boss asks you to manage different versions of each document or not? The larger your website is, the more you need a *Content Management Systems* [http://en.wikipedia.org/wiki/Content_management_system] (CMS) which tackles all these issues.

These CMS solutions are sold by different vendors and each vendor provides its own API for interfacing the proprietary content repository. The developers have to deal with this and need to learn the vendor-specific API. If in the future you wish to switch to a different vendor, everything will be different and you will have a new implementation, a new interface, etc.

JCR provides a unique java interface for interacting with both text and binary data, for dealing with any kind and amount of meta data your documents might have. JCR supplies methods for storing, updating, deleting and retrieving your data, independent of the fact if this data is stored in a RDBMS, in a file system or as an XML document - you just don't need to care about. The JCR interface is also defined as classes and methods for searching, versioning, access control, locking, and observation.

Furthermore, an export and import functionality is specified so that a switch to a different vendor is always possible.

What does eXo do?

eXo fully complies a JCR standard *JSR 170* [http://jcp.org/en/jsr/detail?id=170]; therefore with eXo JCR you can use a vendor-independent API. It means that you could switch any time to a different vendor. Using the standard lowers your lifecycle cost and reduces your long term risk.

Of course eXo does not only offer JCR, but also the complete solution for ECM (Enterprise Content Management) and for WCM (Web Content Management).

Further Reading

In order to further understand the theory of JCR and the API, please refer to some external documents about this standard:

- Roy T. Fielding, *JSR 170 Overview: Standardizing the Content Repository Interface* [http://www.day.com/content/dam/day/whitepapers/JSR_170_White_Paper.pdf] (March 13, 2005)
- Benjamin Mestrallet, Tuan Nguyen, Gennady Azarenkov, Francois Moron and Brice Revenant *eXo Platform v2, Portal, JCR, ECM, Groupware and Business Intelligence.* [http://www.theserverside.com/tt/articles/article.tss?l=eXoPlatform2] (January 2006)

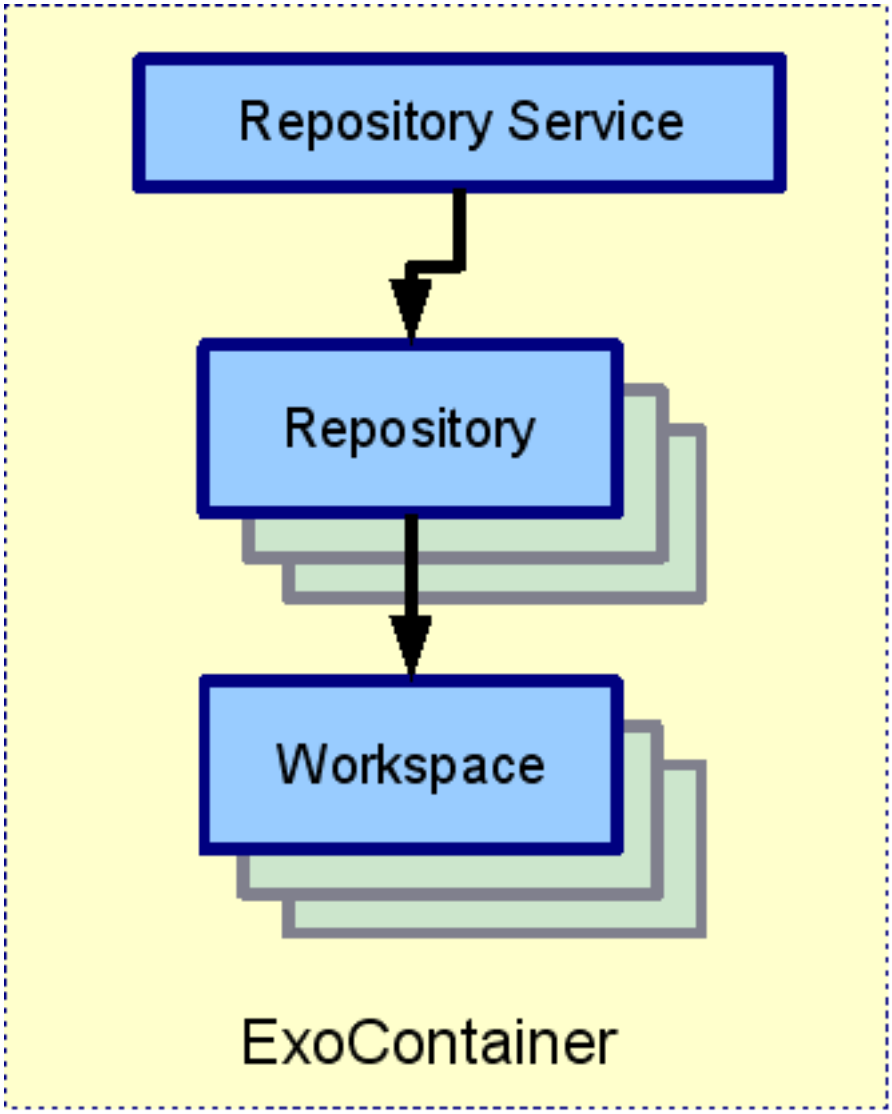
eXo JCR Implementation

Related Documents

Access Control Configuration, Export Import Implementation, External Value Storages, JDBC Data Container config, Locking, Multilanguage support, Node types and Namespaces, Repository and Workspace management, Repository container life cycle, Workspace, Persistence Storage Workspace, SimpleDB storage

How it works

eXo Repository Service is a standard eXo service and is a registered IoC component, i.e. can be deployed in some eXo Containers (see *Service configuration* for details). The relationships between components are shown in the picture below:



eXo Container: some subclasses of org.exoplatform.container.ExoContainer (usually org.exoplatform.container.StandaloneContainer or org.exoplatform.container.PortalContainer) that holds a reference to Repository Service.

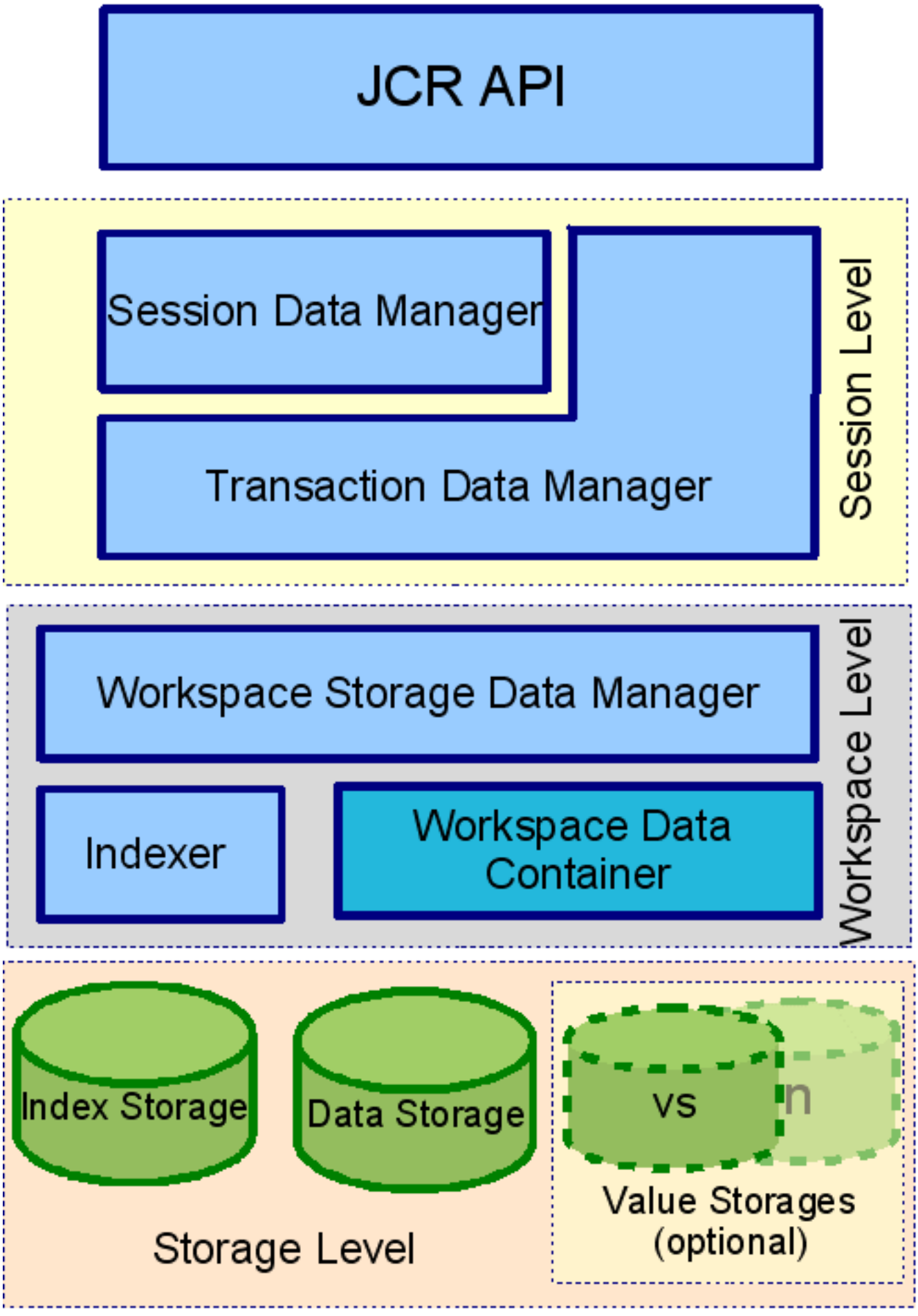
- **Repository Service:** contains information about repositories. eXo JCR is able to manage many Repositories.
- **Repository:** Implementation of javax.jcr.Repository. It holds references to one or more Workspace(s).
- **Workspace:** Container of a single rooted tree of Items. (Note that here it is not exactly the same as javax.jcr.Workspace as it is not a per Session object).

Usual JCR application use case includes two initial steps:

- Obtaining Repository object by getting **Repository Service** from the current eXo Container (eXo "native" way) or via JNDI lookup if eXo repository is bound to the naming context using (see *Service configuration* for details).
- Creating javax.jcr.Session object that calls Repository.login(..).

Workspace Data Model

The following diagram explains which components of eXo JCR implementation are used in a data flow to perform operations specified in JCR API


JCR API
Session Data Manager
Transaction Data Manager
Session Level
Workspace Storage Data Manager
Indexer
Workspace Data Container
Workspace Level
Index Storage
Data Storage
vs
n
Value Storages (optional)
Storage Level

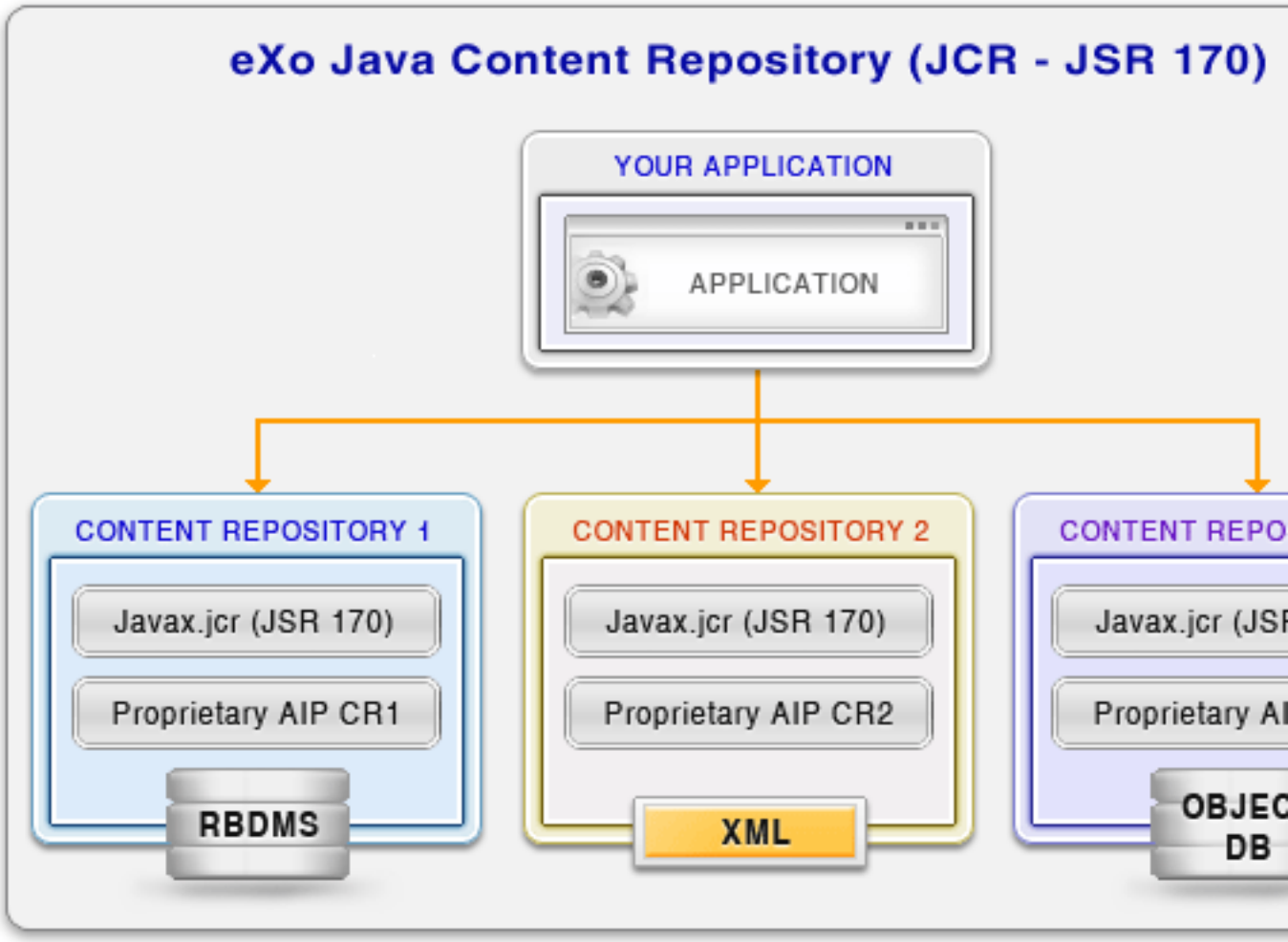
The Workspace Data Model can be split into 4 levels by data isolation and value from the JCR model point of view.

- eXo JCR core implements **JCR API** interfaces, such as Item, Node, Property. It contains JCR "logical" view on stored data.
- **Session Level**: isolates transient data viewable inside one JCR Session and interacts with API level using eXo JCR internal API.
- **Session Data Manager**: maintains transient session data. With data access/ modification/ validation logic, it contains Modified Items Storage to hold the data changed between subsequent save() calling and Session Items Cache.
- **Transaction Data Manager**: maintains session data between save() and transaction commit/ rollback if the current session is part of a transaction.
- **Workspace Level**: operates for particular workspace shared data. It contains per-Workspace objects
- **Workspace Storage Data Manager:** maintains workspace data, including final validation, events firing, caching.
- **Workspace Data Container**: implements physical data storage. It allows different types of backend (like RDB, FS files, etc) to be used as a storage for JCR data. With the main Data Container, other storages for persisted Property Values can be configured and used.
- **Indexer:** maintains workspace data indexing for further queries.
- **Storage Level**: Persistent storages for:
  - JCR Data
  - Indexes (Apache Lucene)
  - Values (e.g., for BLOBs) if different from the main Data Container

Advantages of eXo JCR

Advantages for application developers:

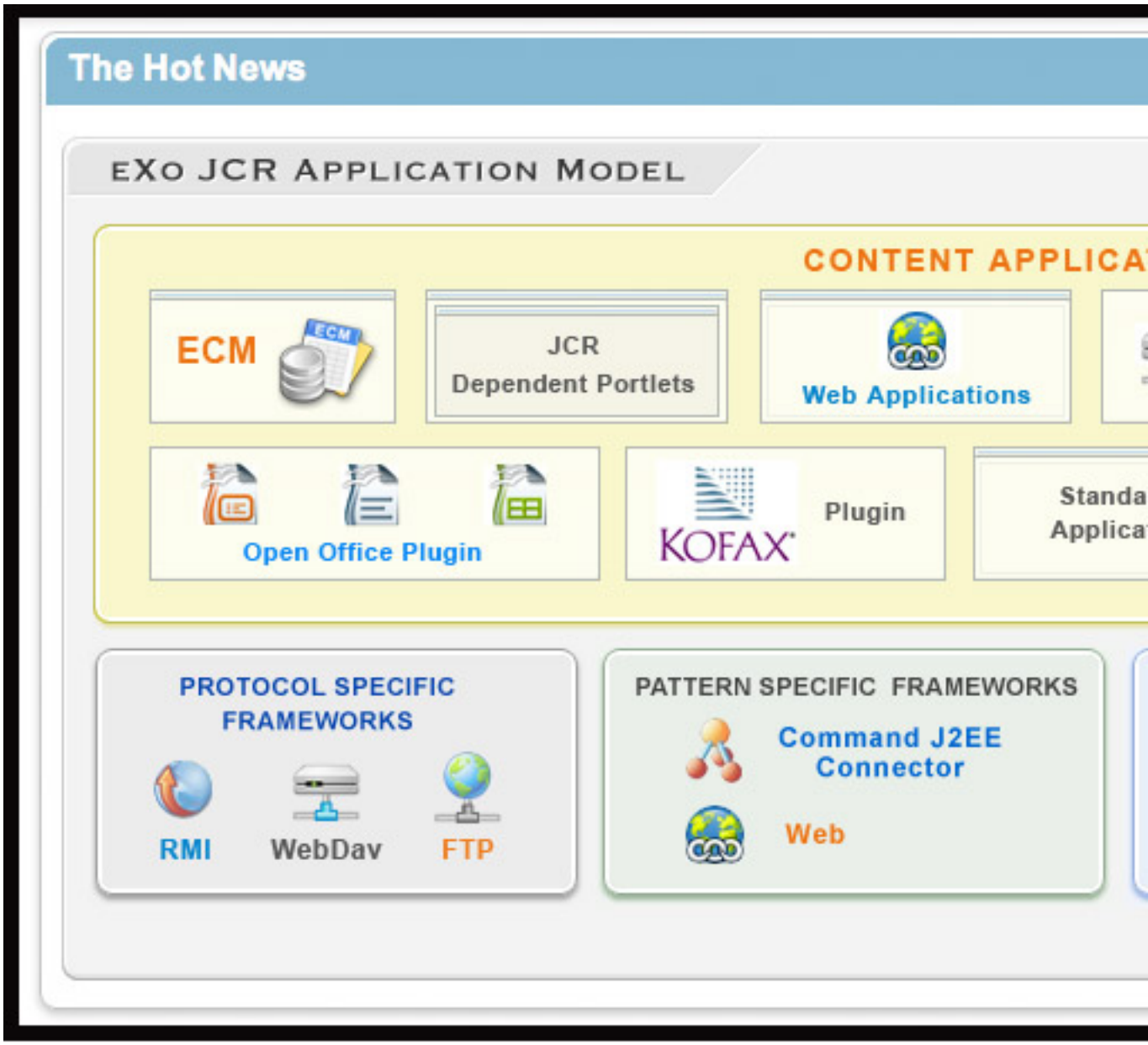
- Data repository and application are isolated from each other so an application developer should not learn the details of particular data storage's interfaces, but can need to concentrate on business logic of a particular application built on the top of JCR.
- Repositories can be simply exchanged between different applications without changing the applications themselves. This is the matter of the repository configuration.
- Data storage types/ versions can be changed and also, different types of data storages can be combined in one repository data model (of course, the complexity and work of building interfaces between the repository and its data storage don't disappear but these changes are isolated in the repository and thus manageable from the point of view of the customer).



Advantages for managers

- Using a standardized repository for content management reduces the risk of dependence on a particular software vendor and proprietary API.
- Costs for maintaining and developing a content repository based custom application is significantly lower than developing and supporting your own interfaces and maintaining your own data repository applications (staff can be trained once, it is possible to take help from the community and the third party consulters).
- Thanks to flexible layered JCR API (see below), it is possible to fit the legacy storage subsystem into new interfaces and decrease the costs and the risk of losing data.

An extension to the API exists as we can see in the following layer schema.



Compatibility Levels

Introduction

The Java Content Repository specification JSR-170 has been split into two compliance levels as well as a set of optional features. Level 1 defines a read-only repository, level 2 defines methods for writing content and bidirectional interaction with the repository.

eXo JCR supports JSR-170 level 1 and level 2 and all optional features. The recent JSR-283 is not yet supported.

Level 1

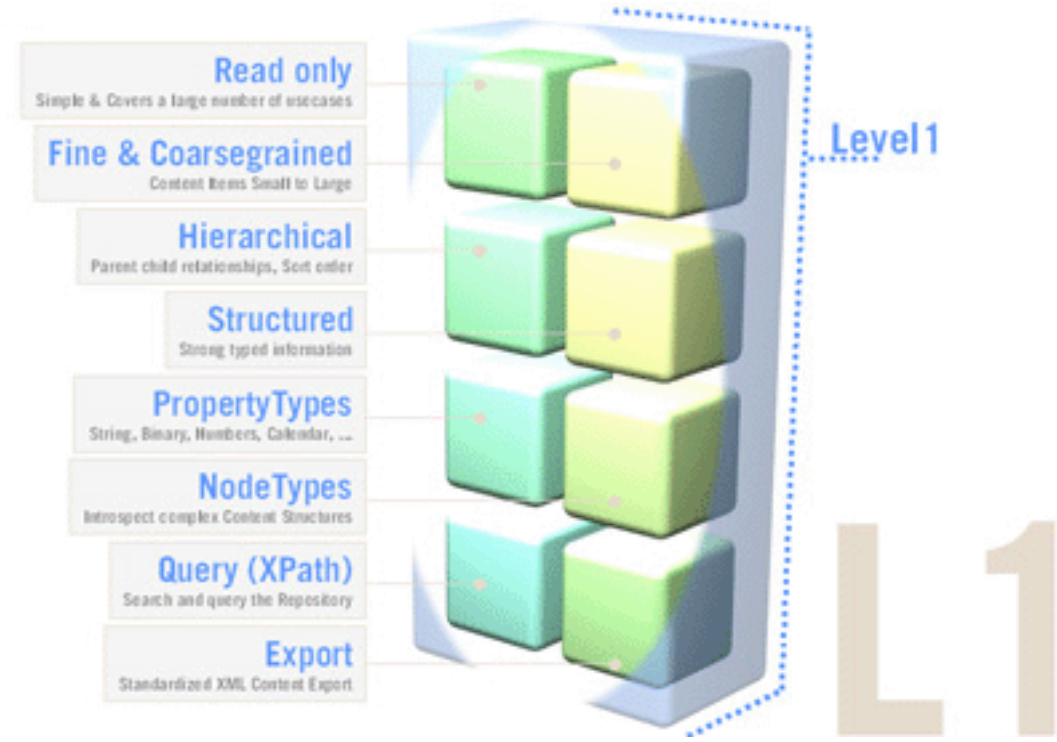
Level 1 includes read-only functionality for very simple repositories. It is useful to port an existing data repository and convert it to a more advanced form step by step. JCR uses a well-known Session abstraction to access the repository data (similar to the sessions we have in OS, web, etc).

The features of level 1:

- Initiating a session calling login method with the name of desired workspace and client credentials. It involves some security mechanisms (JAAS) to authenticate the client and in case the client is authorized to use the data from a particular workspace, he can retrieve the session with a workspace tied to it.
- Using the obtained session, the client can retrieve data (items) by traversing the tree, directly accessing a particular item (requesting path or UUID) or traversing the query result. So an application developer can choose the "best" form depending on the content structure and desired operation.
- Reading property values. All content of a repository is ultimately accessed through properties and stored in property values of predefined types (Boolean, Binary Data, Double, Long, String) and special types Name, Reference, and Path. It is possible to read property value without knowing its real name as a primary item.
- Export to XML. Repository supports two XML/JCR data model mappings: system and doc view. The system view provides complete XML serialization without loss of information and is somewhat difficult for a human to read. In contrast, the document view is well readable but does not completely reflect the state of repository, it is used for Xpath queries.
- Query facility with Xpath syntax. Xpath, originally developed for XML, suits the JCR data model as well because the JCR data model is very close to XML's one. It is applied to JCR as it would be applied to the document view of the serialized repository content, returning a table of property names and content matching the query.
- Discovery of available node types. Every node should have only one primary node type that defines names, types and other characteristics of child nodes and properties. It also can have

one or more mixin data types that defines additional characteristics. Level 1 provides methods for discovering available in repository node types and node types of a concrete node.

- Transient namespace remapping. Item name can have prefix, delimited by a single ':' (colon) character that indicates the namespace of this name. It is patterned after XML namespaces, prefix is mapped to URI to minimize names collisions. In Level 1, a prefix can be temporary overridden by another prefix in the scope of a session.



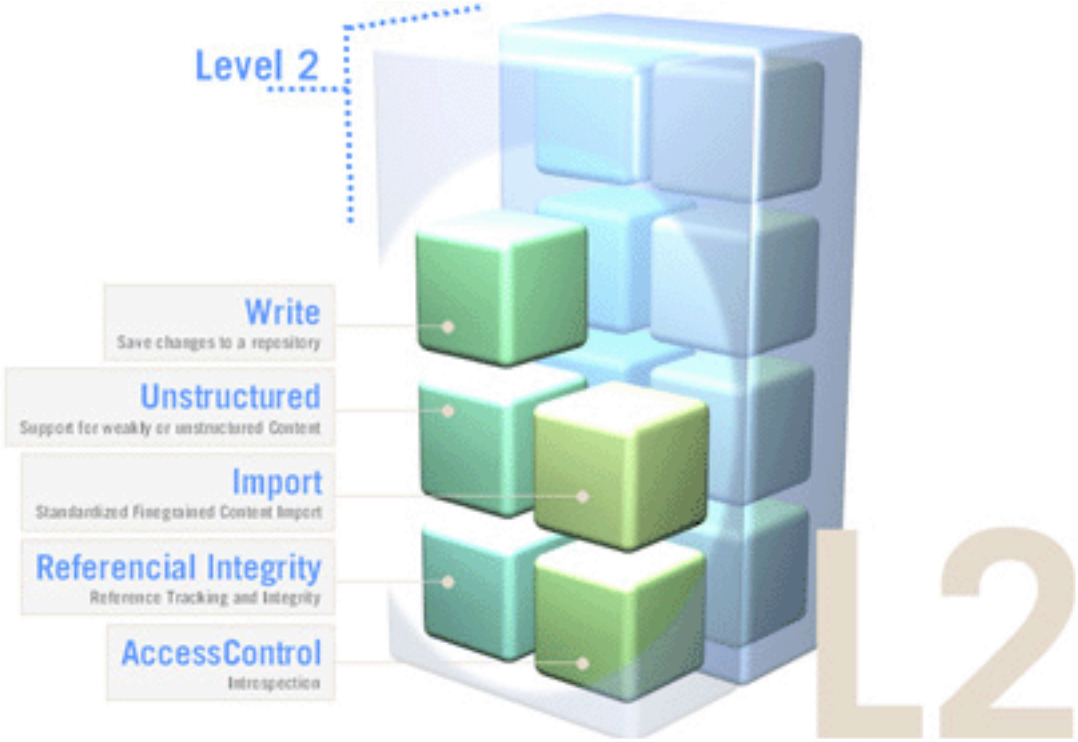
Level 2

JCR level 2 includes reading/ writing content functionality, importing other sources and managing content definition and structuring using extensible node types.

In addition to the features of the Level 1, it also supports the following major features:

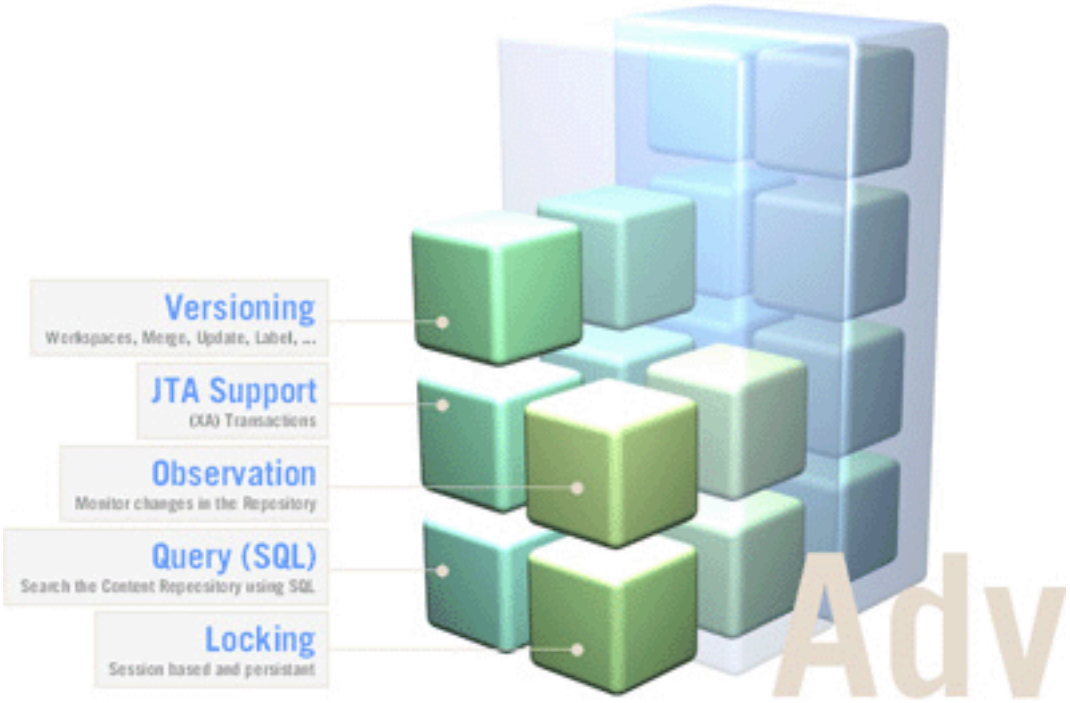
- Adding, moving, copying and removing items inside workspace and moving, copying and cloning items between workspaces. The client can also compare the persisted state of an item with its unsaved states and either save the new state or discard it.
- Modifying and writing value of properties. Property types are checked and can be converted to the defined format.
- Importing XML document into the repository as a tree of nodes and properties. If the XML document is an export of JCR system view, the content of repository can be completely restored. If this is not the case, the document is interpreted as a document view and the import procedure builds a tree of JCR nodes and properties that matches the tree structure of the XML document.

- Assigning node types to nodes. The primary node type is assigned when adding a node. This can be done automatically based on the parent node type definition and mixin node types.
- Persistent namespaces changes. Adding, changing and removing namespaces stored in the namespace registry, excluding built-in namespaces required by JCR.



Optional features

On the top of Level 1 or Level 2, a number of optional features are defined for a more advanced repository functionality. This includes functions such as Versioning, (JTA) Transactions, Query using SQL, Explicit Locking and Content Observation. eXo JCR supports all optional features.


Versioning
Workspaces, Merge, Update, Label, ...
JTA Support
(XA) Transactions
Observation
Monitor changes in the Repository
Query (SQL)
Search the Content Repository using SQL
Locking
Session based and persistant
Adv

Using JCR

1 Using eXo JCR in an application

Obtaining a Repository object

A javax.jcr.Repository object can be obtained by:

- Using the eXo Container "native" mechanism. All Repositories are kept with a single RepositoryService component. So it can be obtained from eXo Container, described as the following:

```
RepositoryService repositoryService = (RepositoryService)
container.getComponentInstanceOfType(RepositoryService.class);
Repository repository = repositoryService.getRepository("repositoryName");
```

- Using the eXo Container "native" mechanism with a thread local saved "current" repository (especially if you plan to use a single repository which covers more than 90% of use cases)

```
// set current repository at initial time
RepositoryService repositoryService = (RepositoryService)
container.getComponentInstanceOfType(RepositoryService.class);
repositoryService.setCurrentRepositoryName("repositoryName");
....
// retrieve and use this repository
Repository repository = repositoryService.getCurrentRepository();
```

- Using JNDI as specified in JSR-170. This way you have to configure the reference (see eXo *JNDI Naming configuration*)

```
Context ctx = new InitialContext();
Repository repository =(Repository) ctx.lookup("repositoryName");
```

JCR Session common considerations

- Remember that javax.jcr.Session is not a thread safe object. **Never try to share it between threads**.

- Do not use **System session** from the **user** related code because a system session has **unlimited rights**. Call ManageableRepository.getSystemSession() from **process** related code only.
- Call Session.logout() explicitly to **release resources** assigned to the session.
- When designing your application, take care of the Session policy inside your application. Two **strategies** are possible: **Stateless** (Session per business request) and **Stateful** (Session per User) or some mix.

JCR Application practices

Simplifying the management of a multi-workspace application

(one-shot logout for all opened sessions)

Use org.exoplatform.services.jcr.ext.common.SessionProvider which is responsible for caching/ obtaining your JCR Sessions and closing all opened sessions at once.

```
public class SessionProvider {

 /**
  * Creates a SessionProvider for a certain identity
  * @param cred
  */
 public SessionProvider(Credentials cred)

 /**
  * Gets the session from internal cache or creates and caches a new one
  */
 public Session getSession(String workspaceName, ManageableRepository repository)
   throws LoginException, NoSuchWorkspaceException, RepositoryException

 /**
  * Calls a logout() method for all cached sessions
  */
 public void close()

 /**
  * a Helper for creating a System session provider
  * @return System session
  */
 public static SessionProvider createSystemProvider()

 /**
```

```
   * a Helper for creating an Anonimous session provider
   * @return System session
   */
  public static SessionProvider createAnonimProvider()

}
```

The SessionProvider is per-request or per-user object, depending on your policy. Create it with your application before performing JCR operations, use it to obtain the Sessions and close at the end of an application session(request). See the following example:

```
// (1) obtain current javax.jcr.Credentials, for example get it from AuthenticationService
Credentials cred = ....

// (2) create SessionProvider for current user
SessionProvider sessionProvider = new SessionProvider(ConversationState.getCurrent());

// NOTE: for creating an Anonymous or System Session use  the corresponding static
 SessionProvider.create...() method
// Get appropriate Repository as described in "Obtaining Repository object" section for example
ManageableRepository repository = (ManageableRepository) ctx.lookup("repositoryName");

// get an appropriate workspace's session
Session session = sessionProvider.getSession("workspaceName", repository);

 .........
// your JCR code
 .........

 // Close the session provider
 sessionProvider.close();
```

Reusing SessionProvider

As shown above, creating the SessionProvider involves multiple steps and you may not want to repeat them each time you need to get a JCR session. In order to avoid all this plumbing code, we provide the SessionProviderService whose goal is to help you to get a SessionProvider object.

The org.exoplatform.services.jcr.ext.app.SessionProviderService interface is defined as follows:

```
public interface SessionProviderService {
  void setSessionProvider(Object key, SessionProvider sessionProvider);
```

```
  SessionProvider getSessionProvider(Object key);
  void removeSessionProvider(Object key);
}
```

Using this service is pretty straightforward, the main contract of an implemented component is getting a SessionProvider by key. eXo provides two implementations :

Table 6.1. SessionProvider implementations

Implementation	Description	Typical Use
org.exoplatform.services.jcr.ext.app.MapStoredSessionProviderService	per-user style : keeps objects in a Map	per-user. The usual practice uses a user's name or Credentials as a key.
org.exoplatform.services.jcr.ext.app.ThreadLocalSessionProviderService	per-request style : keeps a single SessionProvider in a static ThreadLocal variable	Always use null for the key.

For any implementation, your code should follow the following sequence :

- Call SessionProviderService.setSessionProvider(Object key, SessionProvider sessionProvider) at the beginning of a business request for Stateless application or application's session for Statefull policy.
- Call SessionProviderService.getSessionProvider(Object key) for obtaining a SessionProvider object
- Call SessionProviderService.removeSessionProvider(Object key) at the end of a business request for Stateless application or application's session for Statefull policy.

JCR Extensions

JCR Service Extensions

Concept

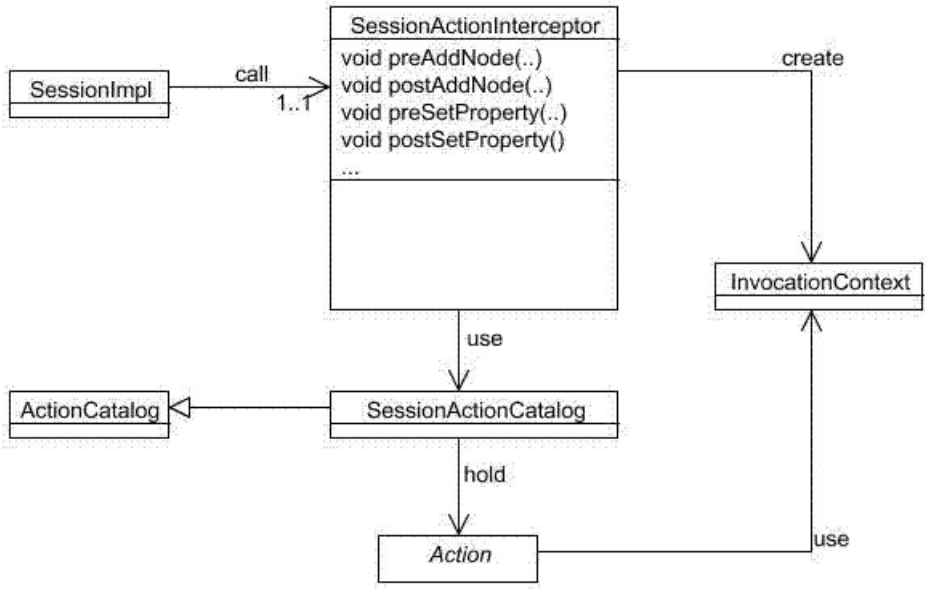
eXo JCR supports **observation** (JSR-170 8.3), which enables applications to register interest in events that describe changes to a workspace, and then monitor and respond to those events. The standard observation feature allows dispatching events when **persistent change** to the workspace is made.

eXo JCR also offers a proprietary **Extension Action** which dispatches and fires an event upon each **transient session level change**, performed by a client. In other words, the event is triggered when a client's program invokes some updating methods in a session or a workspace (such as: Session.addNode(), Session.setProperty(), Workspace.move() etc.

One important recommendation should be applied for an extension action implementation. Each action will add its own execution time to standard JCR methods (Session.addNode(), Session.setProperty(), Workspace.move() etc.) execution time. As a consequence, it's necessary to minimize Action.execute(Context) body execution time.

To make the rule, you can use the dedicated Thread in Action.execute(Context) body for a custom logic. But if your application logic requires the action to add items to a created/updated item and you save these changes immediately after the JCR API method call is returned, the suggestion with Thread is not applicable for you in this case.

Implementation



Interceptor framework class diagram

Configuration

Add a **SessionActionCatalog** service and an appropriate **AddActionsPlugin** (see the example below) configuration to your eXo Container configuration. As usual, the plugin can be configured as in-component-place, which is the case for a Standalone Container or externally, which is a usual case for Root/Portal Container configuration).

Each Action entry is exposed as org.exoplatform.services.jcr.impl.ext.action. **ActionConfiguration** of actions collection of org.exoplatform.services.jcr.impl.ext.action.AddActionsPlugin$ActionsConfig (see an example below). The mandatory field named **actionClassName** is the fully qualified name of org.exoplatform.services.command.action.Action implementation - the command will be launched in case the current event matches the **criteria**. All other fields are criteria. The criteria are *AND*ed together. In other words, for a particular item to be listened to, it must meet ALL the criteria:

* **workspace**: the comma delimited (ORed) list of workspaces

* **eventTypes**: a comma delimited (ORed) **list of event names** (see below) to be listened to. This is the only mandatory field, others are optional and if they are missing they are interpreted as ANY.

* **path** - a comma delimited (ORed) list of **item absolute paths** (or within its subtree if **isDeep** is **true**, which is the default value)

* **nodeTypes** - a comma delimited (ORed) list of the **current NodeType**. Since version 1.6.1 JCR supports the functionalities of nodeType and parentNodeType. This parameter has different semantics, depending on the type of the current item and the operation performed. If the **current item** is a **property** it means the **parent node type**. If the **current item** is a **node,** the semantic depends on the event type: ** **add node event**: the node type of the newly added node. ** **add mixin event**: the newly added mixing node type of the current node. ** **remove mixin event** the removed mixin type of the current node. ** **other events**: the already assigned NodeType(s) of the current node (can be both primary and mixin).

NOTE: the list of fields can be extended.

NOTE2: no spaces between list elements.

NOTE3: isDeep=false means **node, node properties and child nodes**.

The list of supported Event names: **addNode, addProperty, changeProperty, removeProperty, removeNode, addMixin, removeMixin, lock, unlock, checkin, checkout, read.**

```
<component>
  <type>org.exoplatform.services.jcr.impl.ext.action.SessionActionCatalog</type>
  <component-plugins>
    <component-plugin>
      <name>addActions</name>
      <set-method>addPlugin</set-method>
```

```
        <type>org.exoplatform.services.jcr.impl.ext.action.AddActionsPlugin</type>
        <description>add actions plugin</description>
        <init-params>
          <object-param>
            <name>actions</name>
                                                                                <object
 type="org.exoplatform.services.jcr.impl.ext.action.AddActionsPlugin$ActionsConfig">
              <field  name="actions">
                <collection type="java.util.ArrayList">
                  <value>
                    <object type="org.exoplatform.services.jcr.impl.ext.action.ActionConfiguration">
                      <field  name="eventTypes"><string>addNode,removeNode</string></field>
                      <field  name="path"><string>/test,/exo:test</string></field>
                      <field  name="isDeep"><boolean>true</boolean></field>
                    <field  name="nodeTypes"><string>nt:file,nt:folder,mix:lockable</string></field>
                      <!-- field  name="workspace"><string>backup</string></field -->
                                                                                  <field
        name="actionClassName"><string>org.exoplatform.services.jcr.ext.DummyAction</string></
field>
                    </object>
                  </value>
                </collection>
              </field>
            </object>
          </object-param>
        </init-params>
      </component-plugin>
    </component-plugins>
</component>
```

Related Pages

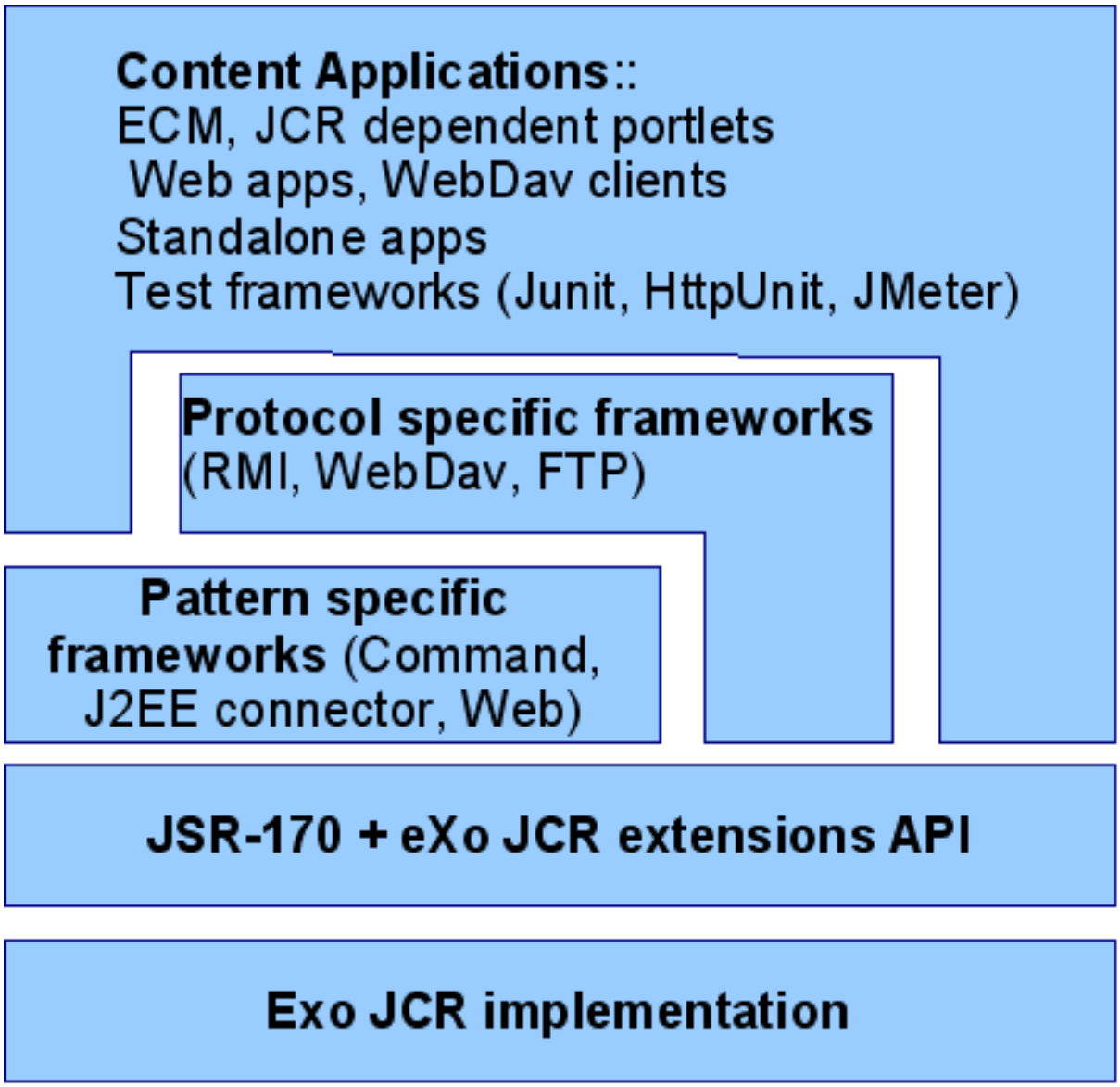
Access Control extension

Registry Service

REST Groovy services

eXo JCR Application Model

The following is a picture about the interaction between Applications and JCR:



Every Content (JCR) dependent application interacts with eXo JCR via JSR-170 and eXo JCR API extension (mostly for administration) directly or using some intermediate Framework (Neither Application nor Framework should ever rely on Implementation directly!)

Content Application: all applications may use JCR as a data storage. Some of them are generic and completely decoupled from JCR API as interaction protocol hides Content storage nature (like WebDav client), some partially decoupled (like Command framework based), meaning that they do not use JCR API directly, and some (most part) use JSR-170 directly.

Frameworks is a special kind of JCR client that acts as an intermediate level between Content Repository and End Client Application. There are **Protocol** (WebDav, RMI or FTP servers

for example) and **Pattern** (Command, Web(servlet), J2EE connector) specific Frameworks. It is possible to build a multi-layered (in framework sense) JCR application, for example Web application uses Web framework that uses Command framework underneath.

NodeType Registration

eXo JCR implementation supports two ways of Nodetypes registration:

- From a NodeTypeValue POJO
- From an XML document (stream)

Interfaces and methods

ExtendedNodeTypeManager

The ExtendedNodeTypeManager (from JCR 1.11) interface provides the following methods related to registering node types:

```
public static final int IGNORE_IF_EXISTS  = 0;

public static final int FAIL_IF_EXISTS    = 2;

public static final int REPLACE_IF_EXISTS = 4;


/**
 * Registers node type using value object.
 */
void registerNodeType(NodeTypeValue nodeTypeValue, int alreadyExistsBehaviour) throws
 RepositoryException;

/**
 * Registers all node types using XML binding value objects from xml stream.
 */
void registerNodeTypes(InputStream xml, int alreadyExistsBehaviour) throws
 RepositoryException;

/**
 * Return <code>NodeTypeValue</code> for a given nodetype name. Used for
 * nodetype update. Value can be edited and registered via
  * <code>registerNodeType(NodeTypeValue nodeTypeValue, int alreadyExistsBehaviour)</
code>
 */
NodeTypeValue getNodeTypeValue(String ntName) throws NoSuchNodeTypeException,
 RepositoryException;
```

```
/**
 * Registers or updates the specified <code>Collection</code> of
 * <code>NodeTypeValue</code> objects.
 */
public NodeTypeIterator registerNodeTypes(Collection<NodeTypeValue> values,
                                          int alreadyExistsBehaviour) throws
 UnsupportedRepositoryOperationException,
                                                RepositoryException;

/**
 * Unregisters the specified node type.
 */
public void unregisterNodeType(String name) throws
 UnsupportedRepositoryOperationException,
                                NoSuchNodeTypeException,
                                RepositoryException;

/**
 * Unregisters the specified set of node types.<p/> Used to unregister a set
 * of node types with mutual dependencies.
 */
public void unregisterNodeTypes(String[] names) throws
 UnsupportedRepositoryOperationException,
                                NoSuchNodeTypeException,
                                RepositoryException;
```

NodeTypeValue

The NodeTypeValue interface represents a simple container structure used to define node types which are then registered through the ExtendedNodeTypeManager.registerNodeType method. The implementation of this interface does not contain any validation logic.

```
/**
 * @return Returns the declaredSupertypeNames.
 */
public List<String> getDeclaredSupertypeNames();

/**
 * @param declaredSupertypeNames
 *The declaredSupertypeNames to set.
 */
public void setDeclaredSupertypeNames(List<String> declaredSupertypeNames);
```

```
/**
 * @return Returns the mixin.
 */
public boolean isMixin();

/**
 * @param mixin
 *The mixin to set.
 */
public void setMixin(boolean mixin);

/**
 * @return Returns the name.
 */
public String getName();

/**
 * @param name
 *The name to set.
 */
public void setName(String name);

/**
 * @return Returns the orderableChild.
 */
public boolean isOrderableChild();

/**
 * @param orderableChild
 *The orderableChild to set.
 */
public void setOrderableChild(boolean orderableChild);

/**
 * @return Returns the primaryItemName.
 */
public String getPrimaryItemName();

/**
 * @param primaryItemName
 *The primaryItemName to set.
 */
public void setPrimaryItemName(String primaryItemName);
```

```
/**
 * @return Returns the declaredChildNodeDefinitionNames.
 */
public List<NodeDefinitionValue> getDeclaredChildNodeDefinitionValues();

/**
 * @param declaredChildNodeDefinitionNames
 *The declaredChildNodeDefinitionNames to set.
 */
public void setDeclaredChildNodeDefinitionValues(List<NodeDefinitionValue>
 declaredChildNodeDefinitionValues);

/**
 * @return Returns the declaredPropertyDefinitionNames.
 */
public List<PropertyDefinitionValue> getDeclaredPropertyDefinitionValues();

/**
 * @param declaredPropertyDefinitionNames
 *The declaredPropertyDefinitionNames to set.
 */
public void setDeclaredPropertyDefinitionValues(List<PropertyDefinitionValue>
 declaredPropertyDefinitionValues);
```

NodeDefinitionValue

The NodeDefinitionValue interface extends ItemDefinitionValue with the addition of writing methods, enabling the characteristics of a child node definition to be set, after that the NodeDefinitionValue is added to a NodeTypeValue.

```
/**
 * @return Returns the declaredSupertypeNames.
 */
public List<String> getDeclaredSupertypeNames();

/**
 * @param declaredSupertypeNames
 *The declaredSupertypeNames to set.
 */
public void setDeclaredSupertypeNames(List<String> declaredSupertypeNames);

/**
 * @return Returns the mixin.
```

```
 */
public boolean isMixin();

/**
 * @param mixin
 *The mixin to set.
 */
public void setMixin(boolean mixin);

/**
 * @return Returns the name.
 */
public String getName();

/**
 * @param name
 *The name to set.
 */
public void setName(String name);

/**
 * @return Returns the orderableChild.
 */
public boolean isOrderableChild();

/**
 * @param orderableChild
 *The orderableChild to set.
 */
public void setOrderableChild(boolean orderableChild);

/**
 * @return Returns the primaryItemName.
 */
public String getPrimaryItemName();

/**
 * @param primaryItemName
 *The primaryItemName to set.
 */
public void setPrimaryItemName(String primaryItemName);

/**
 * @return Returns the declaredChildNodeDefinitionNames.
```

```
 */
public List<NodeDefinitionValue> getDeclaredChildNodeDefinitionValues();

/**
 * @param declaredChildNodeDefinitionNames
 *The declaredChildNodeDefinitionNames to set.
 */
public void setDeclaredChildNodeDefinitionValues(List<NodeDefinitionValue>
 declaredChildNodeDefinitionValues);

/**
 * @return Returns the declaredPropertyDefinitionNames.
 */
public List<PropertyDefinitionValue> getDeclaredPropertyDefinitionValues();

/**
 * @param declaredPropertyDefinitionNames
 *The declaredPropertyDefinitionNames to set.
 */
public void setDeclaredPropertyDefinitionValues(List<PropertyDefinitionValue>
 declaredPropertyDefinitionValues);
```

PropertyDefinitionValue

The PropertyDefinitionValue interface extends ItemDefinitionValue with the addition of writing methods, enabling the characteristics of a child property definition to be set, after that the PropertyDefinitionValue is added to a NodeTypeValue.

```
/**
 * @return Returns the defaultValues.
 */
public List<String> getDefaultValueStrings();

/**
 * @param defaultValues The defaultValues to set.
 */
public void setDefaultValueStrings(List<String> defaultValues);

/**
 * @return Returns the multiple.
 */
public boolean isMultiple();
```

```
/**
 * @param multiple The multiple to set.
 */
public void setMultiple(boolean multiple);

/**
 * @return Returns the requiredType.
 */
public int getRequiredType();

/**
 * @param requiredType The requiredType to set.
 */
public void setRequiredType(int requiredType);

/**
 * @return Returns the valueConstraints.
 */
public List<String> getValueConstraints();

/**
 * @param valueConstraints The valueConstraints to set.
 */
public void setValueConstraints(List<String> valueConstraints);
```

ItemDefinitionValue

```
 /**
 * @return Returns the autoCreate.
 */
public boolean isAutoCreate();

/**
 * @param autoCreate The autoCreate to set.
 */
public void setAutoCreate(boolean autoCreate);

/**
 * @return Returns the mandatory.
 */
public boolean isMandatory();
```

```
/**
 * @param mandatory The mandatory to set.
 */
public void setMandatory(boolean mandatory);

/**
 * @return Returns the name.
 */
public String getName();

/**
 * @param name The name to set.
 */
public void setName(String name);

/**
 * @return Returns the onVersion.
 */
public int getOnVersion();

/**
 * @param onVersion The onVersion to set.
 */
public void setOnVersion(int onVersion);

/**
 * @return Returns the readOnly.
 */
public boolean isReadOnly();

/**
 * @param readOnly The readOnly to set.
 */
public void setReadOnly(boolean readOnly);
```

Node type registration

eXo JCR implementation supports various methods of the node-type registration.

Run time registration from xml file.

```
ExtendedNodeTypeManager nodeTypeManager = (ExtendedNodeTypeManager)
 session.getWorkspace()
```

```
                                        .getNodeTypeManager();
InputStream is = MyClass.class.getResourceAsStream("mynodetypes.xml");
nodeTypeManager.registerNodeTypes(is,ExtendedNodeTypeManager.IGNORE_IF_EXISTS );
```

Run time registration using NodeTypeValue.

```
ExtendedNodeTypeManager nodeTypeManager = (ExtendedNodeTypeManager)
 session.getWorkspace()
                                        .getNodeTypeManager();
NodeTypeValue testNValue = new NodeTypeValue();

List<String> superType = new ArrayList<String>();
superType.add("nt:base");
testNValue.setName("exo:myNodeType");
testNValue.setPrimaryItemName("");
testNValue.setDeclaredSupertypeNames(superType);
List<PropertyDefinitionValue> props = new ArrayList<PropertyDefinitionValue>();
props.add(new PropertyDefinitionValue("*",
                        false,
                        false,
                        1,
                        false,
                        new ArrayList<String>(),
                        false,
                        0,
                        new ArrayList<String>()));
testNValue.setDeclaredPropertyDefinitionValues(props);

nodeTypeManager.registerNodeType(testNValue,
 ExtendedNodeTypeManager.FAIL_IF_EXISTS);
```

Changing existing node type

If you want to replace existing node type definition, you should pass ExtendedNodeTypeManager.REPLACE_IF_EXISTS as a second parameter for the method ExtendedNodeTypeManager.registerNodeType.

```
ExtendedNodeTypeManager nodeTypeManager = (ExtendedNodeTypeManager)
 session.getWorkspace()
                                        .getNodeTypeManager();
InputStream is = MyClass.class.getResourceAsStream("mynodetypes.xml");
```

```
.....
nodeTypeManager.registerNodeTypes(is,ExtendedNodeTypeManager.REPLACE_IF_EXISTS
 );
```

Removing node type

Note

Node type is only possibly removed when the repository does not contain this node type.

```
nodeTypeManager.unregisterNodeType("myNodeType");
```

Practical How to

Adding new PropertyDefinition

```
NodeTypeValue                                   myNodeTypeValue                                 =
 nodeTypeManager.getNodeTypeValue(myNodeTypeName);
List<PropertyDefinitionValue> props = new ArrayList<PropertyDefinitionValue>();
props.add(new PropertyDefinitionValue("tt",
                                true,
                                true,
                                1,
                                false,
                                new ArrayList<String>(),
                                false,
                                PropertyType.STRING,
                                new ArrayList<String>()));
myNodeTypeValue.setDeclaredPropertyDefinitionValues(props);

nodeTypeManager.registerNodeType(myNodeTypeValue,
 ExtendedNodeTypeManager.REPLACE_IF_EXISTS);
```

Adding new child NodeDefinition

```
NodeTypeValue                                   myNodeTypeValue                                 =
 nodeTypeManager.getNodeTypeValue(myNodeTypeName);
```

```
List<NodeDefinitionValue> nodes = new ArrayList<NodeDefinitionValue>();
nodes.add(new NodeDefinitionValue("child",
                                  false,
                                  false,
                                  1,
                                  false,
                                  "nt:base",
                                  new ArrayList<String>(),
                                  false));
testNValue.setDeclaredChildNodeDefinitionValues(nodes);

nodeTypeManager.registerNodeType(myNodeTypeValue,
 ExtendedNodeTypeManager.REPLACE_IF_EXISTS);
```

Changing or removing existing PropertyDefinition or child NodeDefinition

Note that the existing data must be consistent before changing or removing a existing definition . JCR **does not allow** you to change the node type in the way in which the existing data would be incompatible with a new node type. But if these changes are needed, you can do it in several phases, consistently changing the node type and the existing data.

For example:

Add a new residual property definition with name "downloadCount" to the existing node type "myNodeType".

There are two limitations that do not allow us to make the task with a single call of registerNodeType method.

- Existing nodes of the type "myNodeType", which does not contain properties "downloadCount" that conflicts with node type what we need.
- Registered node type "myNodeType" will not allow us to add properties "downloadCount" because it has no such specific properties.

To complete the task, we need to make 3 steps:

- Change the existing node type "myNodeType" by adding the mandatory property "downloadCount".
- Add the node type "myNodeType" with the property "downloadCount" to all the existing node types.
- Change the definition of the property "downloadCount" of the node type "myNodeType" to mandatory.

Changing the list of super types

```
NodeTypeValue testNValue = nodeTypeManager.getNodeTypeValue("exo:myNodeType");

List<String> superType  = testNValue.getDeclaredSupertypeNames();
superType.add("mix:versionable");
testNValue.setDeclaredSupertypeNames(superType);

nodeTypeManager.registerNodeType(testNValue,
 ExtendedNodeTypeManager.REPLACE_IF_EXISTS);
```

Registry Service

Concept

The Registry Service is one of the key parts of the infrastructure built around eXo JCR. Each JCR that is based on service, applications, etc may have its own configuration, settings data and other data that have to be stored persistently and used by the appropriate service or application. (We call it "**Consumer**").

The service acts as a centralized collector (Registry) for such data. Naturally, a registry storage is JCR based i.e. stored in some JCR workspace (one per Repository) as an Item tree under **/exo:registry** node.

Despite the fact that the structure of the tree is well defined (see the scheme below), it is not recommended for other services to manipulate data using JCR API directly for better flexibility. So the Registry Service acts as a mediator between a Consumer and its settings.

The proposed structure of the Registry Service storage is divided into 3 logical groups: services, applications and users:

```
exo:registry/          <-- registry "root" (exo:registry)
  exo:services/        <-- service data storage (exo:registryGroup)
    service1/
      Consumer data    (exo:registryEntry)
    ...
  exo:applications/    <-- application data storage (exo:registryGroup)
    app1/
      Consumer data    (exo:registryEntry)
    ...
  exo:users/           <-- user personal data storage (exo:registryGroup)
    user1/
      Consumer data    (exo:registryEntry)
    ...
```

Each upper level eXo Service may store its configuration in eXo Registry. At first, start from xml-config (in jar etc) and then from Registry. In configuration file, you can add force-xml-configuration parameter to component to ignore reading parameters initialization from RegistryService and to use file instead:

```
<value-param>
  <name>force-xml-configuration</name>
  <value>true</value>
```

```
</value-param>
```

The API

The main functionality of the Registry Service is pretty simple and straightforward, it is described in the Registry abstract class as the following:

```
public abstract class Registry {

  /**
   * Returns the Registry object which wraps the Node of the "exo:registry" type
   */
  public abstract RegistryNode getRegistry(SessionProvider sessionProvider)
      throws RepositoryConfigurationException, RepositoryException;

  /**
   * Returns the existing RegistryEntry which wraps the Node of the "exo:registryEntry" type
   */
  public abstract RegistryEntry getEntry(SessionProvider sessionProvider, String groupName,
      String entryName) throws RepositoryException;

  /**
   * Creates a new RegistryEntry
   */
  public abstract void createEntry(SessionProvider sessionProvider,
      String groupName, RegistryEntry entry) throws RepositoryException;

  /**
   * Replaces a RegistryEntry
   */
  public abstract void recreateEntry(SessionProvider sessionProvider,
      String groupName, RegistryEntry entry) throws RepositoryException;

  /**
   * Removes a RegistryEntry
   */
  public abstract void removeEntry(SessionProvider sessionProvider,
      String groupName, String entryName) throws RepositoryException;
```

As you can see it looks like a simple CRUD interface for the RegistryEntry object which wraps registry data for some Consumer as a Registry Entry. The Registry Service itself knows nothing about the wrapping data, it is Consumer's responsibility to manage and use its data in its own way.

To create an Entity Consumer you should know how to serialize the data to some XML structure and then create a RegistryEntry from these data at once or populate them in a RegistryEntry object (using RegistryEntry(String entryName) constructor and then obtain and fill a DOM document).

Example of RegistryService using:

```
RegistryService regService = (RegistryService) container
.getComponentInstanceOfType(RegistryService.class);

RegistryEntry registryEntry = regService.getEntry(sessionProvider,
      RegistryService.EXO_SERVICES, "my-service");

Document doc = registryEntry.getDocument();

String mySetting = getElementsByTagName("tagname").item(index).getTextContent();
 .....
```

Configuration

RegistryService has only one optional properties parameter **locations**. It is used to mention where exo:registry is placed for each repository. The name of each property is interpreted as a repository name and its value as a workspace name (a system workspace by default).

```
<component>
  <type>org.exoplatform.services.jcr.ext.registry.RegistryService</type>
  <init-params>
    <properties-param>
    <name>locations</name>
    <property name="db1" value="ws2"/>
    </properties-param>
  </init-params>
</component>
```

Namespace altering

Since version 1.11, eXo JCR implementation supports namespaces altering.

Adding new namespace

```
ExtendedNamespaceRegistry namespaceRegistry = (ExtendedNamespaceRegistry) workspace.getNamespaceRegistry();
namespaceRegistry.registerNamespace("newMapping", "http://dumb.uri/jcr");
```

Changing existing namespace

```
ExtendedNamespaceRegistry namespaceRegistry = (ExtendedNamespaceRegistry) workspace.getNamespaceRegistry();
namespaceRegistry.registerNamespace("newMapping", "http://dumb.uri/jcr");
namespaceRegistry.registerNamespace("newMapping2", "http://dumb.uri/jcr");
```

Removing existing namespace

```
ExtendedNamespaceRegistry namespaceRegistry = (ExtendedNamespaceRegistry) workspace.getNamespaceRegistry();
namespaceRegistry.registerNamespace("newMapping", "http://dumb.uri/jcr");
namespaceRegistry.unregisterNamespace("newMapping");
```

Node Types and Namespaces

Introduction

Support of node types and namespaces is required by the JSR-170 specification. Beyond the methods required by the specification, eXo JCR has its own API extension for the *Node type registration* as well as the ability to declaratively define node types in the Repository at the start-up time.

Node Types definition

Node type registration extension is declared in org.exoplatform.services.jcr.core.nodetype.ExtendedNodeTypeManager interface

Your custom service can register some neccessary predefined node types at the start-up time. The node definition should be placed in a special XML file (see DTD below) and declared in the service's configuration file thanks to eXo component plugin mechanism, described as follows:

```
<external-component-plugins>
 <target-component>org.exoplatform.services.jcr.RepositoryService</target-component>
    <component-plugin>
      <name>add.nodeType</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.jcr.impl.AddNodeTypePlugin</type>
      <init-params>
        <values-param>
          <name>autoCreatedInNewRepository</name>
          <description>Node types configuration file</description>
          <value>jar:/conf/test/nodetypes-tck.xml</value>
          <value>jar:/conf/test/nodetypes-impl.xml</value>
        </values-param>
  <values-param>
          <name>repo1</name>
          <description>Node types configuration file for repository with name repo1</description>
          <value>jar:/conf/test/nodetypes-test.xml</value>
        </values-param>
  <values-param>
          <name>repo2</name>
          <description>Node types configuration file for repository with name repo2</description>
          <value>jar:/conf/test/nodetypes-test2.xml</value>
        </values-param>
      </init-params>
    </component-plugin>
```

There are two types of registration. The first type is the registration of node types in all created repositories, it is configured in values-param with the name **autoCreatedInNewRepository**. The second type is registration of node types in specified repository and it is configured in values-param with the name of repository.

Node type definition file format:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nodeTypes [
 <!ELEMENT nodeTypes (nodeType)*>
   <!ELEMENT nodeType (supertypes?|propertyDefinitions?|childNodeDefinitions?)>

   <!ATTLIST nodeType
     name CDATA #REQUIRED
     isMixin (true|false) #REQUIRED
     hasOrderableChildNodes (true|false)
     primaryItemName CDATA
   >
   <!ELEMENT supertypes (supertype*)>
   <!ELEMENT supertype (CDATA)>

   <!ELEMENT propertyDefinitions (propertyDefinition*)>

   <!ELEMENT propertyDefinition (valueConstraints?|defaultValues?)>
   <!ATTLIST propertyDefinition
     name CDATA #REQUIRED
      requiredType (String|Date|Path|Name|Reference|Binary|Double|Long|Boolean|undefined)
#REQUIRED
     autoCreated (true|false) #REQUIRED
     mandatory (true|false) #REQUIRED
    onParentVersion (COPY|VERSION|INITIALIZE|COMPUTE|IGNORE|ABORT) #REQUIRED
     protected (true|false) #REQUIRED
     multiple  (true|false) #REQUIRED
   >
  <!-- For example if you need to set ValueConstraints [],
   you have to add an empty element <valueConstraints/>.
   The same order is for other properties like defaultValues, requiredPrimaryTypes etc.
   -->
   <!ELEMENT valueConstraints (valueConstraint*)>
   <!ELEMENT valueConstraint (CDATA)>
   <!ELEMENT defaultValues (defaultValue*)>
   <!ELEMENT defaultValue (CDATA)>

   <!ELEMENT childNodeDefinitions (childNodeDefinition*)>
```

```
    <!ELEMENT childNodeDefinition (requiredPrimaryTypes)>
    <!ATTLIST childNodeDefinition
      name CDATA #REQUIRED
      defaultPrimaryType  CDATA #REQUIRED
      autoCreated (true|false) #REQUIRED
      mandatory (true|false) #REQUIRED
     onParentVersion (COPY|VERSION|INITIALIZE|COMPUTE|IGNORE|ABORT) #REQUIRED
      protected (true|false) #REQUIRED
      sameNameSiblings (true|false) #REQUIRED
    >
    <!ELEMENT requiredPrimaryTypes (requiredPrimaryType+)>
    <!ELEMENT requiredPrimaryType (CDATA)>
]>
```

Namespaces definition

Default namespaces are registered by repository at the start-up time

Your custom service can extend a set of namespaces with some application specific ones, declaring it in service's configuration file thanks to eXo component plugin mechanism, described as follows:

```
    <component-plugin>
      <name>add.namespaces</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.jcr.impl.AddNamespacesPlugin</type>
      <init-params>
        <properties-param>
          <name>namespaces</name>
          <property name="test" value="http://www.test.org/test"/>
        </properties-param>
      </init-params>
    </component-plugin>
```

eXo JCR configuration

Related documents

- *Configuration persister*
- *Search Configuration*
- *JDBC Data Container config*
- *External Value Storages*
- *Workspace Persistence Storage*

Portal and Standalone configuration

Like other eXo services, eXo JCR can be configured and used in the portal or embedded mode (as a service embedded in eXo Portal) and in standalone mode.

In Embedded mode, JCR services are registered in the Portal container and the second option is to use a Standalone container. The main difference between these container types is that the first one is intended to be used in a Portal (Web) environment, while the second one can be used standalone (see the comprehensive page *Service Configuration for Beginners* for more details).

The following setup procedure is used to obtain a Standalone configuration (see more in *Container configuration*):

- Configuration that is set explicitly using StandaloneContainer.addConfigurationURL(String url) or StandaloneContainer.addConfigurationPath(String path) before getInstance()
- Configuration from $base:directory/exo-configuration.xml or $base:directory/conf/exo-configuration.xml file. Where $base:directory is either AS's home directory in case of J2EE AS environment or just the current directory in case of a standalone application.
- /conf/exo-configuration.xml in the current classloader (e.g. war, ear archive)
- Configuration from $service_jar_file/conf/portal/configuration.xml. WARNING: Don't rely on some concrete jar's configuration if you have more than one jar containing conf/portal/configuration.xml file. In this case choosing a configuration is unpredictable.

JCR service configuration looks like:

```
<component>
  <key>org.exoplatform.services.jcr.RepositoryService</key>
  <type>org.exoplatform.services.jcr.impl.RepositoryServiceImpl</type>
</component>
<component>
  <key>org.exoplatform.services.jcr.config.RepositoryServiceConfiguration</key>
```

```
  <type>org.exoplatform.services.jcr.impl.config.RepositoryServiceConfigurationImpl</type>
  <init-params>
    <value-param>
      <name>conf-path</name>
      <description>JCR repositories configuration file</description>
      <value>jar:/conf/standalone/exo-jcr-config.xml</value>
    </value-param>
    <properties-param>
      <name>working-conf</name>
      <description>working-conf</description>
      <property name="source-name" value="jdbcjcr" />
      <property name="dialect" value="hsqldb" />
                                                    <property          name="persister-class-name"
 value="org.exoplatform.services.jcr.impl.config.JDBCConfigurationPersister" />
    </properties-param>
  </init-params>
</component>
```

conf-path : a path to a RepositoryService JCR Configuration.

working-conf : optional; *JCR configuration persister* configuration. If there isn't a working-conf, the persister will be disabled.

JCR Configuration

The Configuration is defined in an XML file (see DTD below).

JCR Service can use multiple **Repositories** and each repository can have multiple **Workspaces**.

From v.1.9 JCR, repositories configuration parameters support human-readable formats of values. They are all case-insensitive:

- Numbers formats: K,KB - kilobytes, M,MB - megabytes, G,GB - gigabytes, T,TB - terabytes. Examples: 100.5 - digit 100.5, 200k - 200 Kbytes, 4m - 4 Mbytes, 1.4G - 1.4 Gbytes, 10T - 10 Tbytes
- Time format endings: ms - milliseconds, m - minutes, h - hours, d - days, w - weeks, if no ending - seconds. Examples: 500ms - 500 milliseconds, 20 - 20 seconds, 30m - 30 minutes, 12h - 12 hours, 5d - 5 days, 4w - 4 weeks.

Repository service configuration (JCR repositories configuration)

Service configuration may be placed in jar:/conf/standalone/exo-jcr-config.xml for standalone mode. For portal mode, it is located in the portal web application portal/WEB-INF/conf/jcr/repository-configuration.xml.

default-repository: The name of a default repository (one returned by RepositoryService.getRepository()).

repositories: The list of repositories.

Repository configuration:

name: The name of a repository.

default-workspace: The name of a workspace obtained using Session's login() or login(Credentials) methods (ones without an explicit workspace name).

system-workspace: The name of workspace where **/jcr:system** node is placed.

security-domain: The name of a security domain for JAAS authentication.

access-control: The name of an access control policy. There can be 3 types: optional - ACL is created on-demand(default), disable - no access control, mandatory - an ACL is created for each added node(not supported yet).

authentication-policy: The name of an authentication policy class.

workspaces: The list of workspaces.

session-max-age: The time after which an idle session will be removed (called logout). If session-max-age is not set up, idle session will never be removed.

lock-remover-max-threads: Number of threads that can serve LockRemover tasks. Default value is 1. Repository may have many workspaces, each workspace have own LockManager. JCR supports Locks with defined lifetime. Such a lock must be removed is it become expired. That is what LockRemovers does. But LockRemovers is not an independent timer-threads, its a task that executed each 30 seconds. Such a task is served by ThreadPoolExecutor which may use different number of threads.

Workspace configuration:

name: The name of a workspace

auto-init-root-nodetype: DEPRECATED in JCR 1.9 (use initializer). The node type for root node initialization.

container: Workspace data container (physical storage) configuration.

initializer: Workspace initializer configuration.

cache: Workspace storage cache configuration.

query-handler: Query handler configuration.

auto-init-permissions: DEPRECATED in JCR 1.9 (use initializer). Default permissions of the root node. It is defined as a set of semicolon-delimited permissions containing a group of

space-delimited identities (user, group, etc, see Organization service documentation for details) and the type of permission. For example, any read; **:/admin read;**:/admin add_node; **:/admin set_property;**:/admin remove means that users from group **admin** have all permissions and other users have only a 'read' permission.

Workspace data container configuration:

class: A workspace data container class name.

properties: The list of properties (name-value pairs) for the concrete Workspace data container.

Table 13.1.

trigger_events_for_descendents_on_rename	indicates if need to trigger events for descendents on rename or not. It allows to increase performance on rename operation but in same time Observation'll not notified, has default value true
lazy-node-iterator-page-size	the page size for lazy iterator. Indicates how many nodes can be retrieved from storage per request. The default value is 100
acl-bloomfilter-false-positive-probability	ACL Bloom-filter desired false positive probability. Range [0..1]. Default value 0.1d. (See the note below)
acl-bloomfilter-elements-number	Expected number of ACL-elements in the Bloom-filter. Default value 1000000. (See the note below)

Note

Bloom filters are not supported by all the cache implementations so far only the inplementation for infinispan supports it.

value-storages: The list of value storage plugins.

Value Storage plugin configuration (for data container):

Note

The value-storage element is optional. If you don't include it, the values will be stored as BLOBs inside the database.

value-storage: Optional value Storage plugin definition.

class: A value storage plugin class name (attribute).

properties: The list of properties (name-value pairs) for a concrete Value Storage plugin.

filters: The list of filters defining conditions when this plugin is applicable.

Initializer configuration (optional):

class: Initializer implementation class.

properties: The list of properties (name-value pairs). Properties are supported.

root-nodetype: The node type for root node initialization.

root-permissions: Default permissions of the root node. It is defined as a set of semicolon-delimited permissions containing a group of space-delimited identities (user, group etc, see Organization service documentation for details) and the type of permission. For example any read; **:/admin read;**:/admin add_node; **:/admin set_property;**:/admin remove means that users from group **admin** have all permissions and other users have only a 'read' permission.

Configurable initializer adds a capability to override workspace initial startup procedure (used for Clustering). Also it replaces workspace element parameters auto-init-root-nodetype and auto-init-permissions with root-nodetype and root-permissions.

Cache configuration:

enabled: If workspace cache is enabled or not.

class: Cache implementation class, optional from 1.9. Default value is. org.exoplatform.services.jcr.impl.dataflow.persistent.LinkedWorkspaceStorageCacheImpl.

Cache can be configured to use concrete implementation of WorkspaceStorageCache interface. JCR core has two implementation to use:

- LinkedWorkspaceStorageCacheImpl - default, with configurable read behavior and statistic.
- WorkspaceStorageCacheImpl - pre 1.9, still can be used.

properties: The list of properties (name-value pairs) for Workspace cache.

max-size: Cache maximum size (maxSize prior to v.1.9).

live-time: Cached item live time (liveTime prior to v.1.9).

From 1.9 LinkedWorkspaceStorageCacheImpl supports additional optional parameters.

statistic-period: Period (time format) of cache statistic thread execution, 5 minutes by default.

statistic-log: If true cache statistic will be printed to default logger (log.info), false by default or not.

statistic-clean: If true cache statistic will be cleaned after was gathered, false by default or not.

cleaner-period: Period of the eldest items remover execution, 20 minutes by default.

blocking-users-count: Number of concurrent users allowed to read cache storage, 0 - unlimited by default.

Query Handler configuration:

class: A Query Handler class name.

properties: The list of properties (name-value pairs) for a Query Handler (indexDir).

Properties and advanced features described in *Search Configuration*.

Lock Manager configuration:

time-out: Time after which the unused global lock will be removed.

persister: A class for storing lock information for future use. For example, remove lock after jcr restart.

path: A lock folder. Each workspace has its own one.

Note

Also see *lock-remover-max-threads [51]* repository configuration parameter.

```
<!ELEMENT repository-service (repositories)>
<!ATTLIST repository-service default-repository NMTOKEN #REQUIRED>
<!ELEMENT repositories (repository)>
<!ELEMENT repository (security-domain,access-control,session-max-age,authentication-policy,workspaces)>
<!ATTLIST repository
  default-workspace NMTOKEN #REQUIRED
  name NMTOKEN #REQUIRED
  system-workspace NMTOKEN #REQUIRED
>
<!ELEMENT security-domain (#PCDATA)>
<!ELEMENT access-control (#PCDATA)>
<!ELEMENT session-max-age (#PCDATA)>
<!ELEMENT authentication-policy (#PCDATA)>
<!ELEMENT workspaces (workspace+)>
<!ELEMENT workspace (container,initializer,cache,query-handler)>
<!ATTLIST workspace name NMTOKEN #REQUIRED>
<!ELEMENT container (properties,value-storages)>
<!ATTLIST container class NMTOKEN #REQUIRED>
<!ELEMENT value-storages (value-storage+)>
<!ELEMENT value-storage (properties,filters)>
<!ATTLIST value-storage class NMTOKEN #REQUIRED>
```

```
<!ELEMENT filters (filter+)>
<!ELEMENT filter EMPTY>
<!ATTLIST filter property-type NMTOKEN #REQUIRED>
<!ELEMENT initializer (properties)>
<!ATTLIST initializer class NMTOKEN #REQUIRED>
<!ELEMENT cache (properties)>
<!ATTLIST cache
  enabled NMTOKEN #REQUIRED
  class NMTOKEN #REQUIRED
>
<!ELEMENT query-handler (properties)>
<!ATTLIST query-handler class NMTOKEN #REQUIRED>
<!ELEMENT access-manager (properties)>
<!ATTLIST access-manager class NMTOKEN #REQUIRED>
<!ELEMENT lock-manager (time-out,persister)>
<!ELEMENT time-out (#PCDATA)>
<!ELEMENT persister (properties)>
<!ELEMENT properties (property+)>
<!ELEMENT property EMPTY>
```

Help application to prohibit the use of closed sessions

Products that use eXo JCR, sometimes missuse it since they continue to use a session that has been closed through a method call on a node, a property or even the session itself. To prevent bad practices we propose three modes which are the folllowing:

1. If the system property *exo.jcr.prohibit.closed.session.usage* has been set to *true*, then a RepositoryException will be thrown any time an application will try to access to a closed session. In the stack trace, you will be able to know the call stack that closes the session.
2. If the system property *exo.jcr.prohibit.closed.session.usage* has not been set and the system property *exo.product.developing* has been set to *true*, then a warning will be logged in the log file with the full stack trace in order to help identifying the root cause of the issue. In the stack trace, you will be able to know the call stack that closes the session.
3. If none of the previous system properties have been set, then we will ignore that the issue and let the application use the closed session as it was possible before without doing anything in order to allow applications to migrate step by step.

Multilanguage support in eXo JCR RDB backend

Introduction

Whenever relational database is used to store multilingual text data of eXo Java Content Repository, we need to adapt configuration in order to support UTF-8 encoding. Here is a short HOWTO instruction for several supported RDBMS with examples.

The configuration file you have to modify: .../webapps/portal/WEB-INF/conf/jcr/repository-configuration.xml

Note

Datasource *jdbcjcr* used in examples can be configured via `InitialContextInitializer` component.

Oracle

In order to run multilanguage JCR on an Oracle backend Unicode encoding for characters set should be applied to the database. Other Oracle globalization parameters don't make any impact. The only property to modify is `NLS_CHARACTERSET`.

We have tested `NLS_CHARACTERSET` = `AL32UTF8` and it works well for many European and Asian languages.

Example of database configuration (used for JCR testing):

```
NLS_LANGUAGE           AMERICAN
NLS_TERRITORY          AMERICA
NLS_CURRENCY           $
NLS_ISO_CURRENCY       AMERICA
NLS_NUMERIC_CHARACTERS .,
NLS_CHARACTERSET       AL32UTF8
NLS_CALENDAR           GREGORIAN
NLS_DATE_FORMAT        DD-MON-RR
NLS_DATE_LANGUAGE      AMERICAN
NLS_SORT               BINARY
NLS_TIME_FORMAT        HH.MI.SSXFF AM
NLS_TIMESTAMP_FORMAT   DD-MON-RR HH.MI.SSXFF AM
NLS_TIME_TZ_FORMAT     HH.MI.SSXFF AM TZR
NLS_TIMESTAMP_TZ_FORMAT  DD-MON-RR HH.MI.SSXFF AM TZR
```

```
NLS_DUAL_CURRENCY       $
NLS_COMP                BINARY
NLS_LENGTH_SEMANTICS    BYTE
NLS_NCHAR_CONV_EXCP     FALSE
NLS_NCHAR_CHARACTERSET  AL16UTF16
```

JCR 1.12.x doesn't use NVARCHAR columns, so that the value of the parameter NLS_NCHAR_CHARACTERSET does not matter for JCR.

Create database with Unicode encoding and use Oracle dialect for the Workspace Container:

```
<workspace name="collaboration">
                                                                    <container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
          <properties>
            <property name="source-name" value="jdbcjcr" />
            <property name="dialect" value="oracle" />
            <property name="multi-db" value="false" />
            <property name="max-buffer-size" value="200k" />
            <property name="swap-directory" value="target/temp/swap/ws" />
          </properties>
        .....
```

DB2

DB2 Universal Database (DB2 UDB) supports *UTF-8 and UTF-16/UCS-2* [http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/c0004821.htm]. When a Unicode database is created, CHAR, VARCHAR, LONG VARCHAR data are stored in UTF-8 form. It's enough for JCR multi-lingual support.

Example of UTF-8 database creation:

```
DB2 CREATE DATABASE dbname USING CODESET UTF-8 TERRITORY US
```

Create database with UTF-8 encoding and use db2 dialect for Workspace Container on DB2 v.9 and higher:

```
<workspace name="collaboration">
```

```
                                                                    <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
        <properties>
          <property name="source-name" value="jdbcjcr" />
          <property name="dialect" value="db2" />
          <property name="multi-db" value="false" />
          <property name="max-buffer-size" value="200k" />
          <property name="swap-directory" value="target/temp/swap/ws" />
        </properties>
      .....
```

Note

For DB2 v.8.x support change the property "dialect" to db2v8.

MySQL

JCR MySQL-backend requires special dialect *MySQL-UTF8* [http://dev.mysql.com/doc/refman/5.0/en/charset-unicode-utf8.html] to be used for internationalization support. But the database default charset should be latin1 to use limited index space effectively (1000 bytes for MyISAM engine, 767 for InnoDB). If database default charset is multibyte, a JCR database initialization error is thrown concerning index creation failure. In other words, JCR can work on any singlebyte default charset of database, with UTF8 supported by MySQL server. But we have tested it only on latin1 database default charset.

Repository configuration, workspace container entry example:

```
<workspace name="collaboration">
                                                                    <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
        <properties>
          <property name="source-name" value="jdbcjcr" />
          <property name="dialect" value="mysql-utf8" />
          <property name="multi-db" value="false" />
          <property name="max-buffer-size" value="200k" />
          <property name="swap-directory" value="target/temp/swap/ws" />
        </properties>
      .....
```

PostgreSQL

On PostgreSQL-backend, multilingual support can be enabled in *different ways* [http://www.postgresql.org/docs/8.3/interactive/charset.html]:

- Using the locale features of the operating system to provide locale-specific collation order, number formatting, translated messages, and other aspects. UTF-8 is widely used on Linux distributions by default, so it can be useful in such case.
- Providing a number of different character sets defined in the PostgreSQL server, including multiple-byte character sets, to support storing text of any languages, and providing character set translation between client and server. We recommend to use UTF-8 database charset, it will allow any-to-any conversations and make this issue transparent for the JCR.

Create database with UTF-8 encoding and use PgSQL dialect for Workspace Container:

```
<workspace name="collaboration">
                                                                                <container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
          <properties>
            <property name="source-name" value="jdbcjcr" />
            <property name="dialect" value="pgsql" />
            <property name="multi-db" value="false" />
            <property name="max-buffer-size" value="200k" />
            <property name="swap-directory" value="target/temp/swap/ws" />
          </properties>
        .....
```

Search Configuration

XML Configuration

JCR index configuration. You can find this file here: `.../portal/WEB-INF/conf/jcr/repository-configuration.xml`

```
<repository-service default-repository="db1">
 <repositories>
  <repository name="db1" system-workspace="ws" default-workspace="ws">
    ....
   <workspaces>
    <workspace name="ws">
    ....
      <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
        <properties>
          <property name="index-dir" value="${java.io.tmpdir}/temp/index/db1/ws" />
                                                  <property name="synonymprovider-class"
value="org.exoplatform.services.jcr.impl.core.query.lucene.PropertiesSynonymProvider" />
          <property name="synonymprovider-config-path" value="/synonyms.properties" />
          <property name="indexing-config-path" value="/indexing-configuration.xml" />
                                                              <property name="query-class"
value="org.exoplatform.services.jcr.impl.core.query.QueryImpl" />
        </properties>
      </query-handler>
     ...
    </workspace>
  </workspaces>
 </repository>
</repositories>
</repository-service>
```

Configuration parameters

Table 15.1.

Parameter	Default	Description	Since
index-dir	none	The location of the index directory. This parameter is mandatory. Up to 1.9,	1.0

Parameter	Default	Description	Since
		this parameter called "indexDir"	
use-compoundfile	true	Advises lucene to use compound files for the index files.	1.9
min-merge-docs	100	Minimum number of nodes in an index until segments are merged.	1.9
volatile-idle-time	3	Idle time in seconds until the volatile index part is moved to a persistent index even though minMergeDocs is not reached.	1.9
max-merge-docs	Integer.MAX_VALUE	Maximum number of nodes in segments that will be merged. The default value changed in JCR 1.9 to Integer.MAX_VALUE.	1.9
merge-factor	10	Determines how often segment indices are merged.	1.9
max-field-length	10000	The number of words that are fulltext indexed at most per property.	1.9
cache-size	1000	Size of the document number cache. This cache maps uuids to lucene document numbers	1.9
force-consistencycheck	false	Runs a consistency check on every startup. If false, a consistency check is only performed when the search index	1.9

Parameter	Default	Description	Since
		detects a prior forced shutdown.	
auto-repair	true	Errors detected by a consistency check are automatically repaired. If false, errors are only written to the log.	1.9
query-class	QueryImpl	Class name that implements the javax.jcr.query.Query interface.This class must also extend from the class: org.exoplatform.services.jcr.impl.core.query.AbstractQueryImpl.	1.9
document-order	true	If true and the query does not contain an 'order by' clause, result nodes will be in document order. For better performance when queries return a lot of nodes set to 'false'.	1.9
result-fetch-size	Integer.MAX_VALUE	The number of results when a query is executed. Default value: Integer.MAX_VALUE (-> all).	1.9
excerptprovider-class	DefaultXMLExcerpt	The name of the class that implements org.exoplatform.services.jcr.impl.core.query.lucene.ExcerptProvide and should be used for the rep:excerpt() function in a query.	1.9
support-highlighting	false	If set to true additional information is stored in the index to support	1.9

Parameter	Default	Description	Since
		highlighting using the rep:excerpt() function.	
synonymprovider-class	none	The name of a class that implements org.exoplatform.services.jcr.impl.core.query.lucene.SynonymProvi The default value is null (-> not set).	1.9
synonymprovider-config-path	none	The path to the synonym provider configuration file. This path interpreted is relative to the path parameter. If there is a path element inside the SearchIndex element, then this path is interpreted and relative to the root path of the path. Whether this parameter is mandatory or not, it depends on the synonym provider implementation. The default value is null (-> not set).	1.9
indexing-configuration-path	none	The path to the indexing configuration file.	1.9
indexing-configuration-class	IndexingConfigurationImp	The name of the class that implements org.exoplatform.services.jcr.impl.core.query.lucene.IndexingConfig	1.9
force-consistencycheck	false	If setting to true, a consistency check is performed, depending on the parameter forceConsistencyCheck. If setting to false, no	1.9

Parameter	Default	Description	Since
		consistency check is performed on startup, even if a redo log had been applied.	
spellchecker-class	none	The name of a class that implements org.exoplatform.services.jcr.impl.core.query.lucene.SpellChecker.	1.9
spellchecker-more-popular	true	If setting true, spellchecker returns only the suggest words that are as frequent or more frequent than the checked word. If setting false, spellchecker returns null (if checked word exit in dictionary), or spellchecker will return most close suggest word.	1.10
spellchecker-min-distance	0.55f	Minimal distance between checked word and proposed suggest word.	1.10
errorlog-size	50(Kb)	The default size of error log file in Kb.	1.9
upgrade-index	false	Allows JCR to convert an existing index into the new format. Also, it is possible to set this property via system property, for example: -Dupgrade-index=true Indexes before JCR 1.12 will not run with JCR 1.12. Hence you have to run an automatic migration: Start JCR with -Dupgrade-	1.12

Parameter	Default	Description	Since
		index=true. The old index format is then converted in the new index format. After the conversion the new format is used. On the next start, you don't need this option anymore. The old index is replaced and a back conversion is not possible - therefore better take a backup of the index before. (Only for migrations from JCR 1.9 and later.)	
analyzer	org.apache.lucene.analysis.standard.StandardAnalyzer	Class name of a lucene analyzer to use for fulltext indexing of text.	1.12

Global Search Index

Global Search Index Configuration

The global search index is configured in the above-mentioned configuration file (`portal/WEB-INF/conf/jcr/repository-configuration.xml`) in the tag "query-handler".

```
<query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
```

In fact, when using Lucene, you should always use the same analyzer for indexing and for querying, otherwise the results are unpredictable. You don't have to worry about this, eXo JCR does this for you automatically. If you don't like the StandardAnalyzer configured by default, just replace it by your own.

If you don't have a handy QueryHandler, you should learn how to create a customized Handler in 5 minutes.

Customized Search Indexes and Analyzers

By default Exo JCR uses the Lucene standard Analyzer to index contents. This analyzer uses some standard filters in the method that analyzes the content:

```
public TokenStream tokenStream(String fieldName, Reader reader) {
   StandardTokenizer tokenStream = new StandardTokenizer(reader, replaceInvalidAcronym);
   tokenStream.setMaxTokenLength(maxTokenLength);
   TokenStream result = new StandardFilter(tokenStream);
   result = new LowerCaseFilter(result);
   result = new StopFilter(result, stopSet);
   return result;
 }
```

- The first one (StandardFilter) removes 's (as 's in "Peter's") from the end of words and removes dots from acronyms.
- The second one (LowerCaseFilter) normalizes token text to lower case.
- The last one (StopFilter) removes stop words from a token stream. The stop set is defined in the analyzer.

For specific cases, you may wish to use additional filters like ISOLatin1AccentFilter, which replaces accented characters in the ISO Latin 1 character set (ISO-8859-1) by their unaccented equivalents.

In order to use a different filter, you have to create a new analyzer, and a new search index to use the analyzer. You put it in a jar, which is deployed with your application.

Creating the filter

The ISOLatin1AccentFilter is not present in the current Lucene version used by eXo. You can use the attached file. You can also create your own filter, the relevant method is

```
public final Token next(final Token reusableToken) throws java.io.IOException
```

which defines how chars are read and used by the filter.

Creating the analyzer

The analyzer has to extends org.apache.lucene.analysis.standard.StandardAnalyzer, and overload the method

```
public TokenStream tokenStream(String fieldName, Reader reader)
```

to put your own filters. You can have a glance at the example analyzer attached to this article.

Creating the search index

Now, we have the analyzer, we have to write the SearchIndex, which will use the analyzer. Your have to extends org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex. You have to write the constructor, to set the right analyzer, and the method

```
public Analyzer getAnalyzer() {
   return MyAnalyzer;
 }
```

to return your analyzer. You can see the attached SearchIndex.

Note

Since 1.12 version, we can set Analyzer directly in configuration. So, creation new SearchIndex only for new Analyzer is redundant.

Configuring your application to use your SearchIndex

In `portal/WEB-INF/conf/jcr/repository-configuration.xml`, you have to replace each

```
<query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
```

by your own class

```
<query-handler class="mypackage.indexation.MySearchIndex">
```

Configure your application to use your Analyzer

In `portal/WEB-INF/conf/jcr/repository-configuration.xml`, you have to add parameter "analyzer" to each query-handler config:

```
<query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
  <properties>
    ...
    <property name="analyzer" value="org.exoplatform.services.jcr.impl.core.MyAnalyzer"/>
    ...
  </properties>
```

```
</query-handler>
```

When you start exo, your SearchIndex will start to index contents with the specified filters.

Indexing Adjustments

IndexingConfiguration

Starting with version 1.9, the default search index implementation in JCR allows you to control which properties of a node are indexed. You also can define different analyzers for different nodes.

The configuration parameter is called indexingConfiguration and per default is not set. This means all properties of a node are indexed.

If you wish to configure the indexing behavior, you need to add a parameter to the query-handler element in your configuration file.

```
<param name="indexing-configuration-path" value="/indexing_configuration.xml"/>
```

Indexing rules

Node Scope Limit

To optimize the index size, you can limit the node scope so that only certain properties of a node type are indexed.

With the below configuration, only properties named Text are indexed for nodes of type nt:unstructured. This configuration also applies to all nodes whose type extends from nt:unstructured.

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM "http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <index-rule nodeType="nt:unstructured">
    <property>Text</property>
  </index-rule>
</configuration>
```

Please note that you have to declare the namespace prefixes in the configuration element that you are using throughout the XML file!

Indexing Boost Value

It is also possible to configure a boost value for the nodes that match the index rule. The default boost value is 1.0. Higher boost values (a reasonable range is 1.0 - 5.0) will yield a higher score value and appear as more relevant.

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM "http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <index-rule nodeType="nt:unstructured"
              boost="2.0">
    <property>Text</property>
  </index-rule>
</configuration>
```

If you do not wish to boost the complete node but only certain properties, you can also provide a boost value for the listed properties:

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM "http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <index-rule nodeType="nt:unstructured">
    <property boost="3.0">Title</property>
    <property boost="1.5">Text</property>
  </index-rule>
</configuration>
```

Conditional Index Rules

You may also add a condition to the index rule and have multiple rules with the same nodeType. The first index rule that matches will apply and all remain ones are ignored:

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM "http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <index-rule nodeType="nt:unstructured"
              boost="2.0"
              condition="@priority = 'high'">
```

```
    <property>Text</property>
  </index-rule>
  <index-rule nodeType="nt:unstructured">
    <property>Text</property>
  </index-rule>
</configuration>
```

In the above example, the first rule only applies if the nt:unstructured node has a priority property with a value 'high'. The condition syntax supports only the equals operator and a string literal.

You may also refer properties in the condition that are not on the current node:

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM "http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <index-rule nodeType="nt:unstructured"
              boost="2.0"
              condition="ancestor::*/@priority = 'high'">
    <property>Text</property>
  </index-rule>
  <index-rule nodeType="nt:unstructured"
              boost="0.5"
              condition="parent::foo/@priority = 'low'">
    <property>Text</property>
  </index-rule>
  <index-rule nodeType="nt:unstructured"
              boost="1.5"
              condition="bar/@priority = 'medium'">
    <property>Text</property>
  </index-rule>
  <index-rule nodeType="nt:unstructured">
    <property>Text</property>
  </index-rule>
</configuration>
```

The indexing configuration also allows you to specify the type of a node in the condition. Please note however that the type match must be exact. It does not consider sub types of the specified node type.

```
<?xml version="1.0"?>
```

```
<!DOCTYPE configuration SYSTEM "http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <index-rule nodeType="nt:unstructured"
              boost="2.0"
              condition="element(*, nt:unstructured)/@priority = 'high'">
    <property>Text</property>
  </index-rule>
</configuration>
```

Exclusion from the Node Scope Index

Per default all configured properties are fulltext indexed if they are of type STRING and included in the node scope index. A node scope search finds normally all nodes of an index. That is, the select jcr:contains(., 'foo') returns all nodes that have a string property containing the word 'foo'. You can exclude explicitly a property from the node scope index:

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM "http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <index-rule nodeType="nt:unstructured">
    <property nodeScopeIndex="false">Text</property>
  </index-rule>
</configuration>
```

Indexing Aggregates

Sometimes it is useful to include the contents of descendant nodes into a single node to easier search on content that is scattered across multiple nodes.

JCR allows you to define indexed aggregates, basing on relative path patterns and primary node types.

The following example creates an indexed aggregate on nt:file that includes the content of the jcr:content node:

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM "http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:jcr="http://www.jcp.org/jcr/1.0"
               xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <aggregate primaryType="nt:file">
```

```
    <include>jcr:content</include>
  </aggregate>
</configuration>
```

You can also restrict the included nodes to a certain type:

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM "http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:jcr="http://www.jcp.org/jcr/1.0"
               xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <aggregate primaryType="nt:file">
    <include primaryType="nt:resource">jcr:content</include>
  </aggregate>
</configuration>
```

You may also use the * to match all child nodes:

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM "http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:jcr="http://www.jcp.org/jcr/1.0"
               xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
                 <aggregate primaryType="nt:file">http://wiki.exoplatform.com/xwiki/bin/edit/JCR/Search+Configuration
    <include primaryType="nt:resource">*</include>
  </aggregate>
</configuration>
```

If you wish to include nodes up to a certain depth below the current node, you can add multiple include elements. E.g. the nt:file node may contain a complete XML document under jcr:content:

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM "http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:jcr="http://www.jcp.org/jcr/1.0"
               xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <aggregate primaryType="nt:file">
    <include>*</include>
    <include>*/*</include>
```

```
    <include>*/*/*</include>
  </aggregate>
</configuration>
```

Property-Level Analyzers

Example

In this configuration section, you define how a property has to be analyzed. If there is an analyzer configuration for a property, this analyzer is used for indexing and searching of this property. For example:

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM "http://www.exoplatform.org/dtd/indexing-configuration-1.0.dtd">
<configuration xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
  <analyzers>
        <analyzer class="org.apache.lucene.analysis.KeywordAnalyzer">
           <property>mytext</property>
        </analyzer>
        <analyzer class="org.apache.lucene.analysis.WhitespaceAnalyzer">
           <property>mytext2</property>
        </analyzer>
  </analyzers>
</configuration>
```

The configuration above means that the property "mytext" for the entire workspace is indexed (and searched) with the Lucene KeywordAnalyzer, and property "mytext2" with the WhitespaceAnalyzer. Using different analyzers for different languages is particularly useful.

The WhitespaceAnalyzer tokenizes a property, the KeywordAnalyzer takes the property as a whole.

Characteristics of Node Scope Searches

When using analyzers, you may encounter an unexpected behavior when searching within a property compared to searching within a node scope. The reason is that the node scope always uses the global analyzer.

Let's suppose that the property "mytext" contains the text : "testing my analyzers" and that you haven't configured any analyzers for the property "mytext" (and not changed the default analyzer in SearchIndex).

If your query is for example:

```
xpath = "//*[jcr:contains(mytext,'analyzer')]"
```

This xpath does not return a hit in the node with the property above and default analyzers.

Also a search on the node scope

```
xpath = "//*[jcr:contains(.,'analyzer')]"
```

won't give a hit. Realize that you can only set specific analyzers on a node property, and that the node scope indexing/analyzing is always done with the globally defined analyzer in the SearchIndex element.

Now, if you change the analyzer used to index the "mytext" property above to

```
<analyzer class="org.apache.lucene.analysis.Analyzer.GermanAnalyzer">
    <property>mytext</property>
</analyzer>
```

and you do the same search again, then for

```
xpath = "//*[jcr:contains(mytext,'analyzer')]"
```

you would get a hit because of the word stemming (analyzers - analyzer).

The other search,

```
xpath = "//*[jcr:contains(.,'analyzer')]"
```

still would not give a result, since the node scope is indexed with the global analyzer, which in this case does not take into account any word stemming.

In conclusion, be aware that when using analyzers for specific properties, you might find a hit in a property for some search text, and you do not find a hit with the same search text in the node scope of the property!

Note

Both index rules and index aggregates influence how content is indexed in JCR. If you change the configuration, the existing content is not automatically re-indexed according to the new rules. You, therefore, have to manually re-index the content when you change the configuration!

Advanced features

eXo JCR supports some advanced features, which are not specified in JSR 170:

- Get a text excerpt with **highlighted words** that matches the query: *ExcerptProvider*.
- Search a term and its **synonyms**: *SynonymSearch*
- Search **similar** nodes: *SimilaritySearch*
- Check **spelling** of a full text query statement: *SpellChecker*
- Define index **aggregates and rules**: IndexingConfiguration.

JCR Configuration persister

Idea

JCR Repository Service uses `org.exoplatform.services.jcr.config.RepositoryServiceConfiguration` component to read its configuration.

```
<component>
   <key>org.exoplatform.services.jcr.config.RepositoryServiceConfiguration</key>
   <type>org.exoplatform.services.jcr.impl.config.RepositoryServiceConfigurationImpl</type>
   <init-params>
     <value-param>
       <name>conf-path</name>
       <description>JCR configuration file</description>
       <value>/conf/standalone/exo-jcr-config.xml</value>
     </value-param>
   </init-params>
 </component>
```

In the example, Repository Service will read the configuration from the file `/conf/standalone/exo-jcr-config.xml`.

But in some cases, it's required to change the configuration on the fly. And know that the new one will be used. Additionally we wish not to modify the original file.

In this case, we have to use the configuration persister feature which allows to store the configuration in different locations.

Usage

On startup `RepositoryServiceConfiguration` component checks if a configuration persister was configured. In that case, it uses the provided `ConfigurationPersister` implementation class to instantiate the persister object.

Configuration with persister:

```
<component>
   <key>org.exoplatform.services.jcr.config.RepositoryServiceConfiguration</key>
   <type>org.exoplatform.services.jcr.impl.config.RepositoryServiceConfigurationImpl</type>
   <init-params>
     <value-param>
       <name>conf-path</name>
```

```
      <description>JCR configuration file</description>
      <value>/conf/standalone/exo-jcr-config.xml</value>
    </value-param>
    <properties-param>
      <name>working-conf</name>
      <description>working-conf</description>
      <property name="source-name" value="jdbcjcr" />
      <property name="dialect" value="mysql" />
                                                <property name="persister-class-name"
value="org.exoplatform.services.jcr.impl.config.JDBCConfigurationPersister" />
    </properties-param>
  </init-params>
 </component>
```

Where:

- *`source-name`*: JNDI source name configured in `InitialContextInitializer` component. (*`sourceName`* prior v.1.9.) Find more in *database configuration*.
- *`dialect`*: SQL dialect which will be used with database from *`source-name`*. Find more in *database configuration*.
- *`persister-class-name`* - class name of `ConfigurationPersister` interface implementation. (*`persisterClassName`* prior v.1.9.)

ConfigurationPersister interface:

```
/**
  * Init persister.
  * Used by RepositoryServiceConfiguration on init.
  * @return - config data stream
  */
 void init(PropertiesParam params) throws RepositoryConfigurationException;

 /**
  * Read config data.
  * @return - config data stream
  */
 InputStream read() throws RepositoryConfigurationException;

 /**
  * Create table, write data.
  * @param confData - config data stream
```

```
 */
 void write(InputStream confData) throws RepositoryConfigurationException;

 /**
  * Tell if the config exists.
  * @return - flag
  */
 boolean hasConfig() throws RepositoryConfigurationException;
```

JCR Core implementation contains a persister which stores the repository configuration in the relational database using JDBC calls - `org.exoplatform.services.jcr.impl.config.JDBCConfigurationPersister`.

The implementation will crate and use table JCR_CONFIG in the provided database.

But the developer can implement his own persister for his particular usecase.

JDBC Data Container Config

Introduction

eXo JCR persistent data container can work in two configuration modes:

- Multi-database: One database for each workspace (used in standalone eXo JCR service mode)
- Single-database: All workspaces persisted in one database (used in embedded eXo JCR service mode, e.g. in eXo portal)

The data container uses the JDBC driver to communicate with the actual database software, i.e. any JDBC-enabled data storage can be used with eXo JCR implementation.

Currently the data container is tested with the following RDBMS:

- MySQL 5.1 MYSQL Connector/J 5.1.8
- PostgresSQL 8.3.7 JDBC4 Driver, Version 8.3-605
- Oracle DB 11g (11.0.1.6), JDBC Driver Oracle 11g R1 (11.1.0.6.0)
- DB2 9,7 IBM Data Server Driver for JDBC and SQLJ (JCC Driver) v.9.1 (fixpack 3a)
- MS SQL Server 2005 SP3 JDBC Driver 2.0
- Sybase 15.0.3 Driver: Sybase jConnect JDBC driver v7 (Build 26502)
- HSQLDB (2.0.0)

Note

Please note, that JCR requires at least READ_COMMITED isolation level and other RDBMS configurations can cause some side-effects and issues. So, please, make sure proper isolation level is configured on database server side.

Note

One more mandatory JCR requirement for underlying databases is a case sensitive collation. Microsoft SQL Server both 2005 and 2008 customers must configure their server with collation corresponding to personal needs and requirements, but obligatorily case sensitive. For more information please refer to Microsoft SQL Server documentation page "Selecting a SQL Server Collation" *here.* [http://msdn.microsoft.com/en-us/library/ms144250.aspx]

Note

Be aware that JCR does not support MyISAM storage engine for the MySQL relational database management system.

Each database software supports ANSI SQL standards but also has its own specifics. So, each database has its own configuration in eXo JCR as a database dialect parameter. If you need a more detailed configuration of the database, it's possible to do that by editing the metadata SQL-script files.

SQL-scripts you can obtain from jar-file exo.jcr.component.core-XXX.XXX.jar:conf/storage/ . They also can be found at SVN *here.* [https://anonsvn.jboss.org/repos/exo-jcr/jcr/trunk/exo.jcr.component.core/src/main/resources/conf/storage/]

In the next two tables correspondence between the scripts and databases is shown.

Table 17.1. Single-database

MySQL DB	jcr-sjdbc.mysql.sql
MySQL DB with utf-8	jcr-sjdbc.mysql-utf8.sql
PostgresSQL	jcr-sjdbc.pqsql.sql
Oracle DB	jcr-sjdbc.ora.sql
DB2 9.7	jcr-sjdbc.db2.sql
MS SQL Server	jcr-sjdbc.mssql.sql
Sybase	jcr-sjdbc.sybase.sql
HSQLDB	jcr-sjdbc.sql

Table 17.2. Multi-database

MySQL DB	jcr-mjdbc.mysql.sql
MySQL DB with utf-8	jcr-mjdbc.mysql-utf8.sql
PostgresSQL	jcr-mjdbc.pqsql.sql
Oracle DB	jcr-mjdbc.ora.sql
DB2 9.7	jcr-mjdbc.db2.sql
MS SQL Server	jcr-mjdbc.mssql.sql
Sybase	jcr-mjdbc.sybase.sql
HSQLDB	jcr-mjdbc.sql

In case the non-ANSI node name is used, it's necessary to use a database with MultiLanguage support[TODO link to MultiLanguage]. Some JDBC drivers need additional parameters for establishing a Unicode friendly connection. E.g. under mysql it's necessary to add an additional parameter for the JDBC driver at the end of JDBC URL. For instance: `jdbc:mysql://exoua.dnsalias.net/portal?characterEncoding=utf8`

There are preconfigured configuration files for HSQLDB. Look for these files in /conf/portal and /conf/standalone folders of the jar-file exo.jcr.component.core-XXX.XXX.jar or source-distribution of eXo JCR implementation.

By default, the configuration files are located in service jars `/conf/portal/configuration.xml` (eXo services including JCR Repository Service) and `exo-jcr-config.xml` (repositories configuration). In eXo portal product, JCR is configured in portal web application `portal/WEB-INF/conf/jcr/jcr-configuration.xml` (JCR Repository Service and related serivces) and repository-configuration.xml (repositories configuration).

Read more about *Repository configuration*.

Multi-database Configuration

You need to configure each workspace in a repository. You may have each one on different remote servers as far as you need.

First of all configure the data containers in the `org.exoplatform.services.naming.InitialContextInitializer` service. It's the JNDI context initializer which registers (binds) naming resources (DataSources) for data containers.

For example (standalone mode, two data containers *`jdbcjcr`* - local HSQLDB, *`jdbcjcr1`* - remote MySQL):

```
<component>
   <key>org.exoplatform.services.naming.InitialContextInitializer</key>
   <type>org.exoplatform.services.naming.InitialContextInitializer</type>
   <component-plugins>
    <component-plugin>
      <name>bind.datasource</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.naming.BindReferencePlugin</type>
      <init-params>
        <value-param>
          <name>bind-name</name>
          <value>jdbcjcr</value>
        </value-param>
        <value-param>
          <name>class-name</name>
          <value>javax.sql.DataSource</value>
        </value-param>
        <value-param>
          <name>factory</name>
          <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
        </value-param>
        <properties-param>
          <name>ref-addresses</name>
          <description>ref-addresses</description>
          <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
```

```
          <property name="url" value="jdbc:hsqldb:file:target/temp/data/portal"/>
          <property name="username" value="sa"/>
          <property name="password" value=""/>
        </properties-param>
      </init-params>
    </component-plugin>
    <component-plugin>
      <name>bind.datasource</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.naming.BindReferencePlugin</type>
      <init-params>
        <value-param>
          <name>bind-name</name>
          <value>jdbcjcr1</value>
        </value-param>
        <value-param>
          <name>class-name</name>
          <value>javax.sql.DataSource</value>
        </value-param>
        <value-param>
          <name>factory</name>
          <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
        </value-param>
        <properties-param>
          <name>ref-addresses</name>
          <description>ref-addresses</description>
          <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
          <property name="url" value="jdbc:mysql://exoua.dnsalias.net/jcr"/>
          <property name="username" value="exoadmin"/>
          <property name="password" value="exo12321"/>
          <property name="maxActive" value="50"/>
          <property name="maxIdle" value="5"/>
          <property name="initialSize" value="5"/>
        </properties-param>
      </init-params>
    </component-plugin>
  <component-plugins>
  <init-params>
    <value-param>
      <name>default-context-factory</name>
      <value>org.exoplatform.services.naming.SimpleContextFactory</value>
    </value-param>
  </init-params>
</component>
```

We configure the database connection parameters:

- *driverClassName*, e.g. "org.hsqldb.jdbcDriver", "com.mysql.jdbc.Driver", "org.postgresql.Driver"
- *url*, e.g. "jdbc:hsqldb:file:target/temp/data/portal", "jdbc:mysql://exoua.dnsalias.net/jcr"
- *username*, e.g. "sa", "exoadmin"
- *password*, e.g. "", "exo12321"

There can be connection pool configuration parameters (org.apache.commons.dbcp.BasicDataSourceFactory):

- *maxActive*, e.g. 50
- *maxIdle*, e.g. 5
- *initialSize*, e.g. 5
- and other according to *Apache DBCP configuration* [http://jakarta.apache.org/commons/dbcp/configuration.html]

When the data container configuration is done, we can configure the repository service. Each workspace will be configured for its own data container.

For example (two workspaces *ws* - jdbcjcr, *ws1* - jdbcjcr1):

```
<workspaces>
  <workspace name="ws" auto-init-root-nodetype="nt:unstructured">
                                                                    <container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
    <properties>
      <property name="source-name" value="jdbcjcr"/>
      <property name="dialect" value="hsqldb"/>
      <property name="multi-db" value="true"/>
      <property name="max-buffer-size" value="200K"/>
      <property name="swap-directory" value="target/temp/swap/ws"/>
    </properties>
    </container>
    <cache enabled="true">
      <properties>
        <property name="max-size" value="10K"/><!-- 10Kbytes -->
        <property name="live-time" value="30m"/><!-- 30 min -->
      </properties>
    </cache>
```

```
  <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
  <properties>
    <property name="index-dir" value="target/temp/index"/>
  </properties>
  </query-handler>
  <lock-manager>
  <time-out>15m</time-out><!-- 15 min -->
  <persister class="org.exoplatform.services.jcr.impl.core.lock.FileSystemLockPersister">
    <properties>
    <property name="path" value="target/temp/lock/ws"/>
    </properties>
  </persister>
  </lock-manager>
 </workspace>
 <workspace name="ws1" auto-init-root-nodetype="nt:unstructured">
                                                                                <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
  <properties>
    <property name="source-name" value="jdbcjcr1"/>
    <property name="dialect" value="mysql"/>
    <property name="multi-db" value="true"/>
    <property name="max-buffer-size" value="200K"/>
    <property name="swap-directory" value="target/temp/swap/ws1"/>
  </properties>
  </container>
  <cache enabled="true">
    <properties>
      <property name="max-size" value="10K"/>
      <property name="live-time" value="5m"/>
    </properties>
  </cache>
  <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
  <properties>
    <property name="index-dir" value="target/temp/index"/>
  </properties>
  </query-handler>
  <lock-manager>
  <time-out>15m</time-out><!-- 15 min -->
  <persister class="org.exoplatform.services.jcr.impl.core.lock.FileSystemLockPersister">
    <properties>
    <property name="path" value="target/temp/lock/ws1"/>
    </properties>
  </persister>
  </lock-manager>
```

```
  </workspace>
</workspaces>
```

- *source-name*: A javax.sql.DataSource name configured in InitialContextInitializer component (was *sourceName* prior JCR 1.9);
- *dialect*: A database dialect, one of "hsqldb", "mysql", "mysql-utf8", "pgsql", "oracle", "oracle-oci", "mssql", "sybase", "derby", "db2", "db2v8" or "auto" for dialect autodetection;
- *multi-db*: Enable multi-database container with this parameter (set value "true");
- *max-buffer-size: A* a threshold (in bytes) after which a javax.jcr.Value content will be swapped to a file in a temporary storage. I.e. swap for pending changes.
- *swap-directory*: A path in the file system used to swap the pending changes.

In this way, we have configured two workspace which will be persisted in two different databases (ws in HSQLDB, ws1 in MySQL).

Note

Starting from v.1.9 *repository configuration* parameters supports human-readable formats of values (e.g. 200K - 200 Kbytes, 30m - 30 minutes etc)

Single-database configuration

It's more simple to configure a single-database data container. We have to configure one naming resource.

For example (embedded mode for *jdbcjcr* data container):

```
<external-component-plugins>
            <target-component>org.exoplatform.services.naming.InitialContextInitializer</target-component>
   <component-plugin>
      <name>bind.datasource</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.naming.BindReferencePlugin</type>
      <init-params>
        <value-param>
          <name>bind-name</name>
          <value>jdbcjcr</value>
        </value-param>
        <value-param>
          <name>class-name</name>
```

```
          <value>javax.sql.DataSource</value>
        </value-param>
        <value-param>
          <name>factory</name>
          <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
        </value-param>
        <properties-param>
          <name>ref-addresses</name>
          <description>ref-addresses</description>
          <property name="driverClassName" value="org.postgresql.Driver"/>
          <property name="url" value="jdbc:postgresql://exoua.dnsalias.net/portal"/>
          <property name="username" value="exoadmin"/>
          <property name="password" value="exo12321"/>
          <property name="maxActive" value="50"/>
          <property name="maxIdle" value="5"/>
          <property name="initialSize" value="5"/>
        </properties-param>
      </init-params>
    </component-plugin>
  </external-component-plugins>
```

And configure repository workspaces in repositories configuration with this one database. Parameter "multi-db" must be switched off (set value "false").

For example (two workspaces *ws* - jdbcjcr, *ws1* - jdbcjcr):

```
<workspaces>
  <workspace name="ws" auto-init-root-nodetype="nt:unstructured">
                                                                    <container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
    <properties>
      <property name="source-name" value="jdbcjcr"/>
      <property name="dialect" value="pgsql"/>
      <property name="multi-db" value="false"/>
      <property name="max-buffer-size" value="200K"/>
      <property name="swap-directory" value="target/temp/swap/ws"/>
    </properties>
    </container>
    <cache enabled="true">
    <properties>
      <property name="max-size" value="10K"/>
      <property name="live-time" value="30m"/>
```

```
    </properties>
    </cache>
    <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
    <properties>
      <property name="index-dir" value="../temp/index"/>
    </properties>
    </query-handler>
    <lock-manager>
    <time-out>15m</time-out>
    <persister class="org.exoplatform.services.jcr.impl.core.lock.FileSystemLockPersister">
      <properties>
      <property name="path" value="target/temp/lock/ws"/>
      </properties>
    </persister>
    </lock-manager>
  </workspace>
  <workspace name="ws1" auto-init-root-nodetype="nt:unstructured">
                                                                                <container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
    <properties>
      <property name="source-name" value="jdbcjcr"/>
      <property name="dialect" value="pgsql"/>
      <property name="multi-db" value="false"/>
      <property name="max-buffer-size" value="200K"/>
      <property name="swap-directory" value="target/temp/swap/ws1"/>
    </properties>
    </container>
    <cache enabled="true">
    <properties>
      <property name="max-size" value="10K"/>
      <property name="live-time" value="5m"/>
    </properties>
    </cache>
    <lock-manager>
    <time-out>15m</time-out>
    <persister class="org.exoplatform.services.jcr.impl.core.lock.FileSystemLockPersister">
      <properties>
      <property name="path" value="target/temp/lock/ws1"/>
      </properties>
    </persister>
    </lock-manager>
  </workspace>
</workspaces>
```

In this way, we have configured two workspaces which will be persisted in one database (PostgreSQL).

Configuration without DataSource

Repository configuration without using of the `javax.sql.DataSource` bounded in JNDI.

This case may be usable if you have a dedicated JDBC driver implementation with special features like XA transactions, statements/connections pooling etc:

- You have to remove the configuration in `InitialContextInitializer` for your database and configure a new one directly in the workspace container.
- Remove parameter "source-name" and add next lines instead. Describe your values for a JDBC driver, database url and username.

Note

But be careful in this case JDBC driver should implement and provide connection pooling. Connection pooling is very recommended for use with JCR to prevent a database overload.

```
<workspace name="ws" auto-init-root-nodetype="nt:unstructured">
                                                                    <container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
   <properties>
     <property name="dialect" value="hsqldb"/>
     <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
     <property name="url" value="jdbc:hsqldb:file:target/temp/data/portal"/>
     <property name="username" value="su"/>
     <property name="password" value=""/>
     ......
```

Dynamic Workspace Creation

Workspaces can be added dynamically during runtime.

This can be performed in two steps:

- Firstly, `ManageableRepository.configWorkspace(WorkspaceEntry wsConfig)` - register a new configuration in RepositoryContainer and create a WorkspaceContainer.
- Secondly, the main step, `ManageableRepository.createWorkspace(String workspaceName)` - creation of a new workspace.

Simple and Complex queries

eXo JCR provides two ways for interact with Database - `JDBCStorageConnection` that uses simple queries and `CQJDBCStorageConection` that uses complex queries for reducing amount of database callings.

Simple queries will be used if you chose `org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer`:

```
<workspaces>
  <workspace name="ws" auto-init-root-nodetype="nt:unstructured">
                                                                    <container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
    ...
  </workspace>
</worksapces>
```

Complex queries will be used if you chose `org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.CQJDBCWorkspaceDataContainer`:

```
<workspaces>
  <workspace name="ws" auto-init-root-nodetype="nt:unstructured">
                                                                    <container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.CQJDBCWorkspaceDataContainer">
    ...
  </workspace>
</worksapces>
```

Why we should use a Complex Queries?

They are optimised to reduce amount of requests to database.
Why we should use a Simple Queries?

Simple queries implemented in way to support as many database dialects as possible.
Simple queries do not use sub queries, left or right joins.

Forse Query Hints

Some databases supports hints to increase query performance (like Oracle, MySQL, etc). eXo JCR have separate Complex Query implementation for Orcale dialect, that uses query hints to increase performance for few important queries.

To enable this option put next configuration property:

```
<workspace name="ws" auto-init-root-nodetype="nt:unstructured">
                                                                    <container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
   <properties>
     <property name="dialect" value="oracle"/>
     <property name="force.query.hints" value="true" />
     ......
```

Query hints enabled by default.

eXo JCR uses query hints only for Complex Query Oracle dialect. For all other dialects this parameter is ignored.

Notes for Microsoft Windows users

The current configuration of eXo JCR uses *Apache DBCP* [http://commons.apache.org/dbcp/] connection pool (`org.apache.commons.dbcp.BasicDataSourceFactory`). It's possible to set a big value for maxActive parameter in `configuration.xml`. That means usage of lots of TCP/IP ports from a client machine inside the pool (i.e. JDBC driver). As a result, the data container can throw exceptions like "Address already in use". To solve this problem, you have to configure the client's machine networking software for the usage of shorter timeouts for opened TCP/IP ports.

Microsoft Windows has *`MaxUserPort`*, *`TcpTimedWaitDelay`* registry keys in the node *`HKEY_LOCAL_MACHINESYSTEMCurrentControlSetServicesTcpipParameters`*, by default these keys are unset, set each one with values like these:

- "TcpTimedWaitDelay"=dword:0000001e, sets TIME_WAIT parameter to 30 seconds, default is 240.
- "MaxUserPort"=dword:00001b58, sets the maximum of open ports to 7000 or higher, default is 5000.

A sample registry file is below:

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters]
"MaxUserPort"=dword:00001b58
"TcpTimedWaitDelay"=dword:0000001e
```

External Value Storages

Introduction

By default JCR Values are stored in the Workspace Data container along with the JCR structure (i.e. Nodes and Properties). eXo JCR offers an additional option of storing JCR Values separately from Workspace Data container, which can be extremely helpful to keep Binary Large Objects (BLOBs) for example.

Value storage configuration is a part of Repository configuration, find more details *there*.

Tree-based storage is recommended for most of cases. If you run an application on Amazon EC2 - the S3 option may be interesting for architecture. Simple 'flat' storage is good in speed of creation/deletion of values, it might be a compromise for a small storages.

Tree File Value Storage

Holds Values in tree-like FileSystem files. path property points to the root directory to store the files.

This is a recommended type of external storage, it can contain large amount of files limited only by disk/volume free space.

A disadvantage is that it's a higher time on Value deletion due to unused tree-nodes remove.

```
<value-storage id="Storage #1"
 class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValueStorage">
    <properties>
      <property name="path" value="data/values"/>
    </properties>
    <filters>
      <filter property-type="Binary" min-value-size="1M"/>
    </filters>
```

Where :

`id`: The value storage unique identifier, used for linking with properties stored in workspace container.
`path`: A location where value files will be stored.

Each file value storage can have the `filter(s)` for incoming values. A filter can match values by property type (property-type), property name (property-name), ancestor path (ancestor-path) and/or size of values stored (min-value-size, in bytes). In code sample, we use a filter with property-type and min-value-size only. I.e. storage for binary values with size greater of 1MB. It's recommended to store properties with large values in file value storage only.

Another example shows a value storage with different locations for large files (min-value-size a 20Mb-sized filter). A value storage uses ORed logic in the process of filter selection. That means the first filter in the list will be asked first and if not matched the next will be called etc. Here a value matches the 20 MB-sized filter min-value-size and will be stored in the path "data/20Mvalues", all other in "data/values".

```
<value-storages>
                                    <value-storage                  id="Storage                  #1"
 class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValueStorage">
   <properties>
    <property name="path" value="data/20Mvalues"/>
   </properties>
   <filters>
    <filter property-type="Binary" min-value-size="20M"/>
   </filters>
  <value-storage>
                                    <value-storage                  id="Storage                  #2"
 class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValueStorage">
   <properties>
    <property name="path" value="data/values"/>
   </properties>
   <filters>
    <filter property-type="Binary" min-value-size="1M"/>
   </filters>
  <value-storage>
<value-storages>
```

Simple File Value Storage

Note

It's not recommended to use in production due to low capacity capabilities on most file systems.

But if you're sure in your file-system or data amount is small it may be useful for you as haves a faster speed of Value removal.

Hold Values in flat FileSystem files. path property points to root directory in order to store files

```
<value-storage                                  id="Storage                                  #1"
 class="org.exoplatform.services.jcr.impl.storage.value.fs.SimpleFileValueStorage">
    <properties>
     <property name="path" value="data/values"/>
```

```
    </properties>
    <filters>
      <filter property-type="Binary" min-value-size="1M"/>
    </filters>
```

Content Addressable Value storage (CAS) support

eXo JCR supports Content-addressable storage feature for Values storing.

Note

Content-addressable storage, also referred to as associative storage and abbreviated CAS, is a mechanism for storing information that can be retrieved based on its content, not its storage location. It is typically used for high-speed storage and retrieval of fixed content, such as documents stored for compliance with government regulations.

Content Addressable Value storage stores unique content once. Different properties (values) with same content will be stored as one data file shared between those values. We can tell the Value content will be shared across some Values in storage and will be stored on one physical file.

Storage size will be decreased for application which governs potentially same data in the content.

Note

For example: if you have 100 different properties containing the same data (e.g. mail attachment), the storage stores only one single file. The file will be shared with all referencing properties.

If property Value changes, it is stored in an additional file. Alternatively the file is shared with other values, pointing to the same content.

The storage calculates Value content address each time the property was changed. CAS write operations are much more expensive compared to the non-CAS storages.

Content address calculation based on java.security.MessageDigest hash computation and tested with MD5 and SHA1 algorithms.

Note

CAS storage works most efficiently on data that does not change often. For data that changes frequently, CAS is not as efficient as location-based addressing.

CAS support can be enabled for Tree and Simple File Value Storage types.

To enable CAS support, just configure it in JCR Repositories configuration as we do for other Value Storages.

```
<workspaces>
      <workspace name="ws">
                                                                                <container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
          <properties>
            <property name="source-name" value="jdbcjcr"/>
            <property name="dialect" value="oracle"/>
            <property name="multi-db" value="false"/>
            <property name="update-storage" value="false"/>
            <property name="max-buffer-size" value="200k"/>
            <property name="swap-directory" value="target/temp/swap/ws"/>
          </properties>
          <value-storages>
<!------------------- here ----------------------->
                                                                    <value-storage       id="ws"
 class="org.exoplatform.services.jcr.impl.storage.value.fs.CASableTreeFileValueStorage">
              <properties>
                <property name="path" value="target/temp/values/ws"/>
                <property name="digest-algo" value="MD5"/>

                                                                <property                    name="vcas-type"

>
                <property name="jdbc-source-name" value="jdbcjcr"/>
                <property name="jdbc-dialect" value="oracle"/>
              </properties>
              <filters>
                <filter property-type="Binary"/>
              </filters>
            </value-storage>
          </value-storages>
```

Properties:

digest-algo: Digest hash algorithm (MD5 and SHA1 were tested);
vcas-type: Value CAS internal data type, JDBC backed is currently implemented org.exoplatform.services.jcr.impl.storage.value.cas.JDBCValueContentAddressStorageImp;l
jdbc-source-name: JDBCValueContentAddressStorageImpl specific parameter, database will be used to save CAS metadata. It's simple to use same as in workspace container;
jdbc-dialect: JDBCValueContentAddressStorageImpl specific parameter, database dialect. It's simple to use the same as in workspace container;

Disabling value storage

JCR allows to disable value storage by adding property into configuration. For interal usage and testing purpose only.

```
<property name="enabled" value="false" />
```

Be careful, all stored values will be unaccessible.

Workspace Data Container

Each Workspace of JCR has its own persistent storage to hold workspace's items data. eXo Content Repository can be configured so that it can use one or more workspaces that are logical units of the repository content. Physical data storage mechanism is configured using mandatory element **container**. The type of container is described in the attribute **class** = fully qualified name of org.exoplatform.services.jcr.storage.WorkspaceDataContainer subclass like

```
<container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
  <properties>
    <property name="source-name" value="jdbcjcr1"/>
    <property name="dialect" value="hsqldb"/>
    <property name="multi-db" value="true"/>
    <property name="max-buffer-size" value="200K"/>
    <property name="swap-directory" value="target/temp/swap/ws"/>
    <property name="lazy-node-iterator-page-size" value="50"/>
    <property name="acl-bloomfilter-false-positive-probability" value="0.1d"/>
    <property name="acl-bloomfilter-elements-number" value="1000000"/>
  </properties>
```

Properties are Container specific parameters:

source-name: JDBC data source name, registered in JDNI by InitialContextInitializer. (**sourceName** prior v.1.9)

dialect: Database dialect, one of "hsqldb", "mysql", "mysql-utf8", "pgsql", "oracle", "oracle-oci", "mssql", "sybase", "derby", "db2", "db2v8"

multi-db: Enable multi-database container with this parameter (if "true").

max-buffer-size: A threshold in bytes, if a value size is greater, then it will be spooled to a temporary file.

swap-directory: A location where the value will be spooled if no value storage is configured but a max-buffer-size is exceeded.

lazy-node-iterator-page-size: "Lazy" child nodes iterator settings. Defines size of page, the number of nodes that are retrieved from persistent storage at once.

acl-bloomfilter-false-positive-probability: ACL Bloom-filter settings. ACL Bloom-filter desired false positive probability. Range [0..1]. Default value 0.1d.

acl-bloomfilter-elements-number: ACL Bloom-filter settings. Expected number of ACL-elements in the Bloom-filter. Default value 1000000.

Note

Bloom filters are not supported by all the cache implementations so far only the inplementation for infinispan supports it.

Bloom-filter used to avoid read nodes that definitely do not have ACL. **acl-bloomfilter-false-positive-probability** and **acl-bloomfilter-elements-number** used to configure such filters. Bloom filters are not supported by all the cache implementations so far only the inplementation for infinispan supports it.

More about Bloom filters you can read here *http://en.wikipedia.org/wiki/Bloom_filter*.

eXo JCR has an RDB (JDBC) based, production ready **Workspace Data Container**.

Workspace Data Container MAY support external storages for javax.jcr.Value (which can be the case for BLOB values for example) using the optional element **value-storages**. Data Container will try to read or write Value using underlying value storage plugin if the filter criteria (see below) match the current property.

```
<value-storages>
                                <value-storage id="Storage #1"
 class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValueStorage">
   <properties>
     <property name="path" value="data/values"/>
   </properties>
   <filters>
    <filter property-type="Binary" min-value-size="1M"/><!-- Values large of 1Mbyte -->
   </filters>
.........
</value-storages>
```

Where **value-storage** is the subclass of org.exoplatform.services.jcr.storage.value.ValueStoragePlugin and **properties** are optional plugin specific parameters.

filters : Each file value storage can have the filter(s) for incoming values. If there are several filter criteria, they all have to match (AND-Condition).

A filter can match values by property type (property-type), property name (property-name), ancestor path (ancestor-path) and/or the size of values stored (min-value-size, e.g. 1M, 4.2G, 100 (bytes)).

In a code sample, we use a filter with property-type and min-value-size only. That means that the storage is only for binary values whose size is greater than 1Mbyte.

It's recommended to store properties with large values in a file value storage only.

REST Services on Groovy

Concept

Starting from version 1.9, JCR Service supports REST services creation on *Groovy script* [http://groovy.codehaus.org].

The feature bases on *RESTful framework* and uses ResourceContainer concept.

Usage

Scripts should extend ResourceContainer and should be stored in JCR as a node of type exo:groovyResourceContainer.

Detailed REST services step-by-step implementation check there *Create REST service step by step*.

Component configuration enables Groovy services loader:

```
<component>
  <type>org.exoplatform.services.jcr.ext.script.groovy.GroovyScript2RestLoader</type>
  <init-params>
    <object-param>
      <name>observation.config</name>
                                                                                <object
 type="org.exoplatform.services.jcr.ext.script.groovy.GroovyScript2RestLoader$ObservationListenerConfiguration">
        <field name="repository">
          <string>repository</string>
        </field>
        <field name="workspaces">
          <collection type="java.util.ArrayList">
            <value>
              <string>collaboration</string>
            </value>
          </collection>
        </field>
      </object>
    </object-param>
  </init-params>
</component>
```

Configuring JBoss AS with eXo JCR in cluster

Launching Cluster

Deploying eXo JCR to JBoss As

To deploy eXo JCR to JBoss, do the following steps:

1. Download the latest version of eXo JCR .ear file distribution.
2. Copy <jcr.ear> into <%jboss_home%/server/default/deploy>
3. Put exo-configuration.xml to the root <%jboss_home%/exo-configuration.xml>
4. Configure JAAS by inserting XML fragment shown below into <%jboss_home%/server/default/conf/login-config.xml>

```
<application-policy name="exo-domain">
  <authentication>
            <login-module   code="org.exoplatform.services.security.j2ee.JbossLoginModule"
 flag="required"></login-module>
  </authentication>
</application-policy>
```

5. Ensure that you use JBossTS *Transaction Service* and JBossCache *Transaction Manager*. Your exo-configuration.xml must contain such parts:

```
<component>
  <key>org.jboss.cache.transaction.TransactionManagerLookup</key>
  <type>org.jboss.cache.GenericTransactionManagerLookup</type>^
</component>

<component>
  <key>org.exoplatform.services.transaction.TransactionService</key>
  <type>org.exoplatform.services.transaction.jbosscache.JBossTransactionsService</type>
  <init-params>
    <value-param>
      <name>timeout</name>
      <value>300</value>
```

```
      </value-param>
    </init-params>
  </component>
```

6. Start server:
   - bin/run.sh for Unix
   - bin/run.bat for Windows
7. Try accessing `http://localhostu:8080/browser` with root/exo as login/password if you have done everything right, you'll get access to repository browser.

Configuring JCR to use external configuration

- To manually configure repository, create a new configuration file (e.g., exo-jcr-configuration.xml). For details, see *JCR Configuration*. Your configuration must look like:

```
<repository-service default-repository="repository1">
  <repositories>
    <repository name="repository1" system-workspace="ws1" default-workspace="ws1">
      <security-domain>exo-domain</security-domain>
      <access-control>optional</access-control>
    <authentication-policy>org.exoplatform.services.jcr.impl.core.access.JAASAuthenticator</
authentication-policy>
      <workspaces>
        <workspace name="ws1">
                                                                    <container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.CQJDBCWorkspaceDataContainer">
            <properties>
              <property name="source-name" value="jdbcjcr" />
              <property name="dialect" value="oracle" />
              <property name="multi-db" value="false" />
              <property name="update-storage" value="false" />
              <property name="max-buffer-size" value="200k" />
              <property name="swap-directory" value="../temp/swap/production" />
            </properties>
            <value-storages>
              see "Value storage configuration" part.
            </value-storages>
          </container>
        <initializer class="org.exoplatform.services.jcr.impl.core.ScratchWorkspaceInitializer">
            <properties>
              <property name="root-nodetype" value="nt:unstructured" />
```

```
            </properties>
          </initializer>

                                                                  <cache
class="org.exoplatform.services.jcr.impl.dataflow.persistent.jbosscache.JBossCacheWorkspaceStorageCache">
              see  "Cache configuration" part.
          </cache>
                                                                <query-handler
class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
              see  "Indexer configuration" part.
          </query-handler>
                                                                <lock-manager
class="org.exoplatform.services.jcr.impl.core.lock.jbosscache.CacheableLockManagerImpl">
              see  "Lock Manager configuration" part.
          </lock-manager>
        </workspace>
        <workspace name="ws2">
              ...
        </workspace>
        <workspace name="wsN">
              ...
        </workspace>
      </workspaces>
    </repository>
  </repositories>
</repository-service>
```

- Then, update RepositoryServiceConfiguration configuration in exo-configuration.xml to use this file:

```
<component>
  <key>org.exoplatform.services.jcr.config.RepositoryServiceConfiguration</key>
  <type>org.exoplatform.services.jcr.impl.config.RepositoryServiceConfigurationImpl</type>
  <init-params>
    <value-param>
      <name>conf-path</name>
      <description>JCR configuration file</description>
      <value>exo-jcr-configuration.xml</value>
    </value-param>
  </init-params>
</component>
```

Requirements

Environment requirements

- Every node of cluster MUST have the same mounted Network File System with the read and write permissions on it.

  "/mnt/tornado" - path to the mounted Network File System (all cluster nodes must use the same NFS).

- Every node of cluster MUST use the same database.

- The same Clusters on different nodes MUST have the same names (e.g., if Indexer cluster in workspace production on the first node has the name "production_indexer_cluster", then indexer clusters in workspace production on all other nodes MUST have the same name "production_indexer_cluster").

Configuration requirements

Configuration of every workspace in repository must contains of such parts:

- Value Storage configuration:

```
<value-storages>
                                                    <value-storage                    id="system"
 class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValueStorage">
     <properties>
       <property name="path" value="/mnt/tornado/temp/values/production" />    <!--path within
 NFS where ValueStorage will hold it's data-->
     </properties>
     <filters>
       <filter property-type="Binary" />
     </filters>
   </value-storage>
</value-storages>
```

- Cache configuration:

```
<cache                                                                    enabled="true"
 class="org.exoplatform.services.jcr.impl.dataflow.persistent.jbosscache.JBossCacheWorkspaceStorageCache">
   <properties>
```

```
        <property  name="jbosscache-configuration"  value="jar:/conf/portal/test-jbosscache-
data.xml" />    <!--   path to JBoss Cache configuration for data storage -->
     <property name="jgroups-configuration" value="jar:/conf/portal/udp-mux.xml" />
     <!--   path to JGroups configuration -->
     <property name="jbosscache-cluster-name" value="JCR_Cluster_cache_production" />
          <!--   JBoss Cache data storage cluster name -->
     <property name="jgroups-multiplexer-stack" value="true" />
  </properties>
</cache>
```

- Indexer configuration:

```
<query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
  <properties>
                                                <property          name="changesfilter-class"

 />
     <property name="index-dir" value="/mnt/tornado/temp/jcrlucenedb/production" />
       <!--   path within NFS where ValueStorage will hold it's data -->
          <property  name="jbosscache-configuration"  value="jar:/conf/portal/test-jbosscache-
indexer.xml" />   <!--   path to JBoss Cache configuration for indexer -->
     <property name="jgroups-configuration" value="jar:/conf/portal/udp-mux.xml" />
       <!--   path to JGroups configuration -->
     <property name="jbosscache-cluster-name" value="JCR_Cluster_indexer_production" />
          <!--   JBoss Cache indexer cluster name -->
     <property name="jgroups-multiplexer-stack" value="true" />
  </properties>
</query-handler>
```

- Lock Manager configuration:

```
<lock-manager
 class="org.exoplatform.services.jcr.impl.core.lock.jbosscache.CacheableLockManagerImpl">
  <properties>
     <property name="time-out" value="15m" />
          <property  name="jbosscache-configuration"  value="jar:/conf/portal/test-jbosscache-
lock.xml" />      <!--   path to JBoss Cache configuration for lock manager -->
     <property name="jgroups-configuration" value="jar:/conf/portal/udp-mux.xml" />
       <!--   path to JGroups configuration -->
     <property name="jgroups-multiplexer-stack" value="true" />
     <property name="jbosscache-cluster-name" value="JCR_Cluster_lock_production" />
          <!--   JBoss Cache locks cluster name -->
```

```
    <property name="jbosscache-cl-cache.jdbc.table.name" value="jcrlocks_production"/>
        <!--  the name of the DB table where lock's data will be stored -->
    <property name="jbosscache-cl-cache.jdbc.table.create" value="true"/>
    <property name="jbosscache-cl-cache.jdbc.table.drop" value="false"/>
                                <property  name="jbosscache-cl-cache.jdbc.table.primarykey"
 value="jcrlocks_production_pk"/>
    <property name="jbosscache-cl-cache.jdbc.fqn.column" value="fqn"/>
    <property name="jbosscache-cl-cache.jdbc.node.column" value="node"/>
    <property name="jbosscache-cl-cache.jdbc.parent.column" value="parent"/>
    <property name="jbosscache-cl-cache.jdbc.datasource" value="jdbcjcr"/>
  </properties>
</lock-manager>
```

JBoss Cache configuration

JBoss cache configuration for indexer, lock manager and data container

Each mentioned components uses instances of JBoss Cache product for caching in clustered environment. So every element has it's own transport and has to be configured in a proper way. As usual, workspaces have similar configuration but with different cluster-names and may-be some other parameters. The simplest way to configure them is to define their own configuration files for each component in each workspace:

```
<property name="jbosscache-configuration" value="conf/standalone/test-jbosscache-lock-db1-ws1.xml" />
```

But if there are few workspaces, configuring them in such a way can be painful and hard-manageable. eXo JCR offers a template-based configuration for JBoss Cache instances. You can have one template for Lock Manager, one for Indexer and one for data container and use them in all the workspaces, defining the map of substitution parameters in a main configuration file. Just simply define ${jbosscache-<parameter name>} inside xml-template and list correct value in JCR configuration file just below "jbosscache-configuration", as shown:

Template:

```
...
<clustering mode="replication" clusterName="${jbosscache-cluster-name}">
  <stateRetrieval timeout="20000" fetchInMemoryState="false" />
...
```

and JCR configuration file:

```
...
<property name="jbosscache-configuration" value="jar:/conf/portal/jbosscache-lock.xml" />
<property name="jbosscache-cluster-name" value="JCR-cluster-locks-db1-ws" />
...
```

JGroups configuration

JGroups is used by JBoss Cache for network communications and transport in a clustered environment. If property "jgroups-configuration" is defined in component configuration, it will be injected into the JBoss Cache instance on startup.

```
<property name="jgroups-configuration" value="your/path/to/modified-udp.xml" />
```

As mentioned above, each component (lock manager, data container and query handler) for each workspace requires its own clustered environment. In other words, they have their own clusters with unique names. By default, each cluster should perform multi-casts on a separate port. This configuration leads to much unnecessary overhead on cluster. That's why JGroups offers multiplexer feature, providing ability to use one single channel for set of clusters. This feature reduces network overheads and increase performance and stability of application. To enable multiplexer stack, you should define appropriate configuration file (upd-mux.xml is pre-shipped one with eXo JCR) and set "jgroups-multiplexer-stack" into "true".

```
<property name="jgroups-configuration" value="jar:/conf/portal/udp-mux.xml" />
<property name="jgroups-multiplexer-stack" value="true" />
```

Allow to share JBoss Cache instances

A JBoss Cache instance is quite resource consuming and by default we will have 3 JBoss Cache instances (one instance for the indexer, one for the lock manager and one for the data container) for each workspace, so if you intend to have a lot of workspaces it could make sense to decide to share one JBoss Cache instance with several cache instances of the same type (i.e. indexer, lock manager or data container). This feature is disabled by default and can be enabled at component configuration level (i.e. indexer configuration, lock manager configuration and/or data container configuration) by setting the property "jbosscache-shareable" to true as below:

```
<property name="jbosscache-shareable" value="true" />
```

Once enabled this feature will allow the JBoss Cache instance used by the component to be re-used by another components of the same type (i.e. indexer, lock manager or data container) with the exact same JBoss Cache configuration (except the eviction configuration that cans be different), which means that all the parameters of type ${jbosscache-<parameter name>} must be identical between the components of same type of different workspaces. In other words, if we use the same values for the parameters of type ${jbosscache-<parameter name>} in each workspace, we will have only 3 JBoss Cache instances (one instance for the indexer, one for the lock manager and one for the data container) used whatever the total amount of workspaces defined.

Shipped JBoss Cache configuration templates

eXo JCR implementation is shipped with ready-to-use JBoss Cache configuration templates for JCR's components. They are situated in application package in /conf/porta/ folder.

Data container template

Data container template is "jbosscache-data.xml":

```
<?xml version="1.0" encoding="UTF-8"?>
<jbosscache                                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="urn:jboss:jbosscache-core:config:3.1">

                          <locking          useLockStriping="false"          concurrencyLevel="50000"
 lockParentForChildInsertRemove="false"
     lockAcquisitionTimeout="20000" />

  <clustering mode="replication" clusterName="${jbosscache-cluster-name}">
     <stateRetrieval timeout="20000" fetchInMemoryState="false" />
     <jgroupsConfig multiplexerStack="jcr.stack" />
     <sync />
  </clustering>

  <!-- Eviction configuration -->
  <eviction wakeUpInterval="5000">
     <default algorithmClass="org.jboss.cache.eviction.LRUAlgorithm"

 actionPolicyClass="org.exoplatform.services.jcr.impl.dataflow.persistent.jbosscache.ParentNodeEvictionActionPoli
        eventQueueSize="1000000">
        <property name="maxNodes" value="1000000" />
        <property name="timeToLive" value="120000" />
     </default>
  </eviction>
</jbosscache>
```

Table 22.1. Template variables

Variable
jbosscache-cluster-name

Lock manager template

It's template name is "jbosscache-lock.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
<jbosscache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="urn:jboss:jbosscache-core:config:3.1">

                        <locking useLockStriping="false" concurrencyLevel="50000"
 lockParentForChildInsertRemove="false"
     lockAcquisitionTimeout="20000" />
   <clustering mode="replication" clusterName="${jbosscache-cluster-name}">
     <stateRetrieval timeout="20000" fetchInMemoryState="false" />
     <jgroupsConfig multiplexerStack="jcr.stack" />
     <sync />
   </clustering>
   <loaders passivation="false" shared="true">
     <preload>
       <node fqn="/" />
     </preload>
                    <loader class="org.jboss.cache.loader.JDBCCacheLoader" async="false"
 fetchPersistentState="false"
       ignoreModifications="false" purgeOnStartup="false">
       <properties>
         cache.jdbc.table.name=${jbosscache-cl-cache.jdbc.table.name}
         cache.jdbc.table.create=${jbosscache-cl-cache.jdbc.table.create}
         cache.jdbc.table.drop=${jbosscache-cl-cache.jdbc.table.drop}
         cache.jdbc.table.primarykey=${jbosscache-cl-cache.jdbc.table.primarykey}
         cache.jdbc.fqn.column=${jbosscache-cl-cache.jdbc.fqn.column}
         cache.jdbc.fqn.type=${jbosscache-cl-cache.jdbc.fqn.type}
         cache.jdbc.node.column=${jbosscache-cl-cache.jdbc.node.column}
         cache.jdbc.node.type=${jbosscache-cl-cache.jdbc.node.type}
         cache.jdbc.parent.column=${jbosscache-cl-cache.jdbc.parent.column}
         cache.jdbc.datasource=${jbosscache-cl-cache.jdbc.datasource}
       </properties>
     </loader>
   </loaders>
</jbosscache>
```

Table 22.2. Template variables

Variable
jbosscache-cluster-name
jbosscache-cl-cache.jdbc.table.name
jbosscache-cl-cache.jdbc.table.create
jbosscache-cl-cache.jdbc.table.drop

Variable
jbosscache-cl-cache.jdbc.table.primarykey
jbosscache-cl-cache.jdbc.fqn.column
jbosscache-cl-cache.jdbc.fqn.type
jbosscache-cl-cache.jdbc.node.column
jbosscache-cl-cache.jdbc.node.type
jbosscache-cl-cache.jdbc.parent.column
jbosscache-cl-cache.jdbc.datasource

Query handler (indexer) template

Have a look at "jbosscache-indexer.xml"

```
<?xml version="1.0" encoding="UTF-8"?>
<jbosscache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="urn:jboss:jbosscache-core:config:3.1">
                        <locking useLockStriping="false" concurrencyLevel="50000"
 lockParentForChildInsertRemove="false"
     lockAcquisitionTimeout="20000" />
   <clustering mode="replication" clusterName="${jbosscache-cluster-name}">
     <stateRetrieval timeout="20000" fetchInMemoryState="false" />
     <jgroupsConfig multiplexerStack="jcr.stack" />
     <sync />
   </clustering>
   <!-- Eviction configuration -->
   <eviction wakeUpInterval="5000">
                            <default algorithmClass="org.jboss.cache.eviction.FIFOAlgorithm"
 eventQueueSize="1000000">
       <property name="maxNodes" value="10000" />
       <property name="minTimeToLive" value="60000" />
     </default>
   </eviction>
</jbosscache>
```

Table 22.3. Template variables

Variable
jbosscache-cluster-name

LockManager configuration

Introduction

What LockManager does?

In general, LockManager stores Lock objects, so it can give a Lock object or can release it, etc.

Also, LockManager is responsible for removing Locks that live too long. This parameter may be configured with "time-out" property.

JCR provides two basic implementations of LockManager:

- `org.exoplatform.services.jcr.impl.core.lock.LockManagerImpl`;
- `org.exoplatform.services.jcr.impl.core.lock.jbosscache.CacheableLockManagerImpl`;

In this article, we will mostly mention about CacheableLockManagerImpl.

You can enable LockManager by adding lock-manager-configuration to workspace-configuration.

For example:

```
<workspace name="ws">
  ...
                                                                <lock-manager
 class="org.exoplatform.services.jcr.impl.core.lock.jbosscache.CacheableLockManagerImpl">
    <properties>
      <property name="time-out" value="15m" />
      ...
    </properties>
  </lock-manager>
  ...
</workspace>
```

LockManagerImpl

LockManagerImpl is a simple implementation of LockManager, and also faster than CacheableLockManager. It stores Lock objects in HashMap and may also persist Locks if LockPersister is configured. LockManagerImpl does not support replication in any way.

See more about LockManager Configuration at *here*.

CacheableLockManagerImpl

CacheableLockManagerImpl stores Lock objects in JBoss-cache, so Locks are replicable and affect on cluster, not only a single node. Also, JBoss-cache has JDBCCacheLoader, so Locks will be stored to the database.

Both of the implementations support to remove Expired Locks. LockRemover separates threads, that periodically ask LockManager to remove Locks that live so long. So, the timeout for LockRemover may be set as follows, the default value is 30m.

```
<properties>
  <property name="time-out" value="10m" />
  ...
</properties>
```

Configuration

Replication requirements are the same for Cache.

You can see a full JCR configuration example at *here*.

Common tips:

- `clusterName` ("jbosscache-cluster-name") must be unique;
- `cache.jdbc.table.name` must be unique per datasource;
- `cache.jdbc.fqn.type` and cache.jdbc.node.type must be configured according to used database;

There are a few ways to configure CacheableLockManagerImpl, and all of them configure JBoss-cache and JDBCCacheLoader.

See *http://community.jboss.org/wiki/JBossCacheJDBCCacheLoader*

Simple JbossCache Configuration

The first one is putting JbossCache configuraion file path to CacheableLockManagerImpl.

Note

This configuration is not so good as you think. Because the repository may contain many workspaces, and each workspace must contain LockManager configuration, and LockManager configuration may contain the JbossCache config file. So, the total

configuration will grow up. But it is useful if we want to have a single LockManager with a special configuration.

Configuration is as follows:

```
<lock-manager
 class="org.exoplatform.services.jcr.impl.core.lock.jbosscache.CacheableLockManagerImpl">
   <properties>
     <property name="time-out" value="15m" />
      <property name="jbosscache-configuration" value="conf/standalone/cluster/test-jbosscache-
lock-config.xml" />
   </properties>
</lock-manager>
```

`test-jbosscache-lock-config.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<jbosscache                                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="urn:jboss:jbosscache-core:config:3.2">

                <locking                 useLockStriping="false"                 concurrencyLevel="50000"
 lockParentForChildInsertRemove="false" lockAcquisitionTimeout="20000" />

 <clustering mode="replication" clusterName="JBoss-Cache-Lock-Cluster_Name">
  <stateRetrieval timeout="20000" fetchInMemoryState="false" nonBlocking="true" />
  <jgroupsConfig>

    <TCP  bind_addr="127.0.0.1"  start_port="9800"  loopback="true"  recv_buf_size="20000000"
 send_buf_size="640000" discard_incompatible_packets="true"
     max_bundle_size="64000"  max_bundle_timeout="30"  use_incoming_packet_handler="true"
 enable_bundling="false" use_send_queues="false" sock_conn_timeout="300"
      skip_suspected_members="true"  use_concurrent_stack="true"  thread_pool.enabled="true"
 thread_pool.min_threads="1" thread_pool.max_threads="25"
                          thread_pool.keep_alive_time="5000"        thread_pool.queue_enabled="false"
 thread_pool.queue_max_size="100" thread_pool.rejection_policy="run"
                               oob_thread_pool.enabled="true"         oob_thread_pool.min_threads="1"
 oob_thread_pool.max_threads="8" oob_thread_pool.keep_alive_time="5000"
                 oob_thread_pool.queue_enabled="false"     oob_thread_pool.queue_max_size="100"
 oob_thread_pool.rejection_policy="run" />
  <MPING timeout="2000" num_initial_members="2" mcast_port="34540" bind_addr="127.0.0.1"
 mcast_addr="224.0.0.1" />
```

```
  <MERGE2 max_interval="30000" min_interval="10000" />
  <FD_SOCK />
  <FD max_tries="5" shun="true" timeout="10000" />
  <VERIFY_SUSPECT timeout="1500" />
                              <pbcast.NAKACK          discard_delivered_msgs="true"          gc_lag="0"
retransmit_timeout="300,600,1200,2400,4800" use_mcast_xmit="false" />
  <UNICAST timeout="300,600,1200,2400,3600" />
  <pbcast.STABLE desired_avg_gossip="50000" max_bytes="400000" stability_delay="1000" />
                 <pbcast.GMS       join_timeout="5000"       print_local_addr="true"       shun="false"
view_ack_collection_timeout="5000" view_bundling="true" />
  <FRAG2 frag_size="60000" />
  <pbcast.STREAMING_STATE_TRANSFER />
 <pbcast.FLUSH timeout="0" />

 </jgroupsConfig

 <sync />
</clustering>

<loaders passivation="false" shared="true">
 <preload>
  <node fqn="/" />
 </preload>
               <loader         class="org.jboss.cache.loader.JDBCCacheLoader"         async="false"
fetchPersistentState="false" ignoreModifications="false" purgeOnStartup="false">
  <properties>
   cache.jdbc.table.name=jcrlocks_ws
   cache.jdbc.table.create=true
   cache.jdbc.table.drop=false
   cache.jdbc.table.primarykey=jcrlocks_ws_pk
   cache.jdbc.fqn.column=fqn
   cache.jdbc.fqn.type=VARCHAR(512)
   cache.jdbc.node.column=node
   cache.jdbc.node.type=<BLOB>
   cache.jdbc.parent.column=parent
   cache.jdbc.datasource=jdbcjcr
  </properties>
 </loader>

 </loaders>

</jbosscache>
```

Configuration requirements:

- <clustering mode="replication" clusterName="JBoss-Cache-Lock-Cluster_Name"> - the cluster name must be unique;
- `cache.jdbc.table.name` must be unique per datasource;
- `cache.jdbc.node.type` and `cache.jdbc.fqn.type` must be configured according to using the database. See *Data Types in Different Databases* .

Template JBossCache Configuration

The second one is using the template JBoss-cache configuration for all LockManagers.

Lock template configuration

`test-jbosscache-lock.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<jbosscache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="urn:jboss:jbosscache-core:config:3.1">

                <locking useLockStriping="false" concurrencyLevel="50000"
lockParentForChildInsertRemove="false"
    lockAcquisitionTimeout="20000" />

  <clustering mode="replication" clusterName="${jbosscache-cluster-name}">
    <stateRetrieval timeout="20000" fetchInMemoryState="false" />
    <jgroupsConfig multiplexerStack="jcr.stack" />
    <sync />
  </clustering>

  <loaders passivation="false" shared="true">
    <!-- All the data of the JCR locks needs to be loaded at startup -->
    <preload>
      <node fqn="/" />
    </preload>
    <!--
    For another cache-loader class you should use another template with
    cache-loader specific parameters
    ->
            <loader class="org.jboss.cache.loader.JDBCCacheLoader" async=q"false"
fetchPersistentState="false"
      ignoreModifications="false" purgeOnStartup="false">
      <properties>
        cache.jdbc.table.name=${jbosscache-cl-cache.jdbc.table.name}
        cache.jdbc.table.create=${jbosscache-cl-cache.jdbc.table.create}
        cache.jdbc.table.drop=${jbosscache-cl-cache.jdbc.table.drop}
```

```
        cache.jdbc.table.primarykey=${jbosscache-cl-cache.jdbc.table.primarykey}
        cache.jdbc.fqn.column=${jbosscache-cl-cache.jdbc.fqn.column}
        cache.jdbc.fqn.type=${jbosscache-cl-cache.jdbc.fqn.type}
        cache.jdbc.node.column=${jbosscache-cl-cache.jdbc.node.column}
        cache.jdbc.node.type=${jbosscache-cl-cache.jdbc.node.type}
        cache.jdbc.parent.column=${jbosscache-cl-cache.jdbc.parent.column}
        cache.jdbc.datasource=${jbosscache-cl-cache.jdbc.datasource}
      </properties>
    </loader>
  </loaders>
</jbosscache>
```

As you see, all configurable parameters are filled by templates and will be replaced by LockManagers configuration parameters:

```
<lock-manager
 class="org.exoplatform.services.jcr.impl.core.lock.jbosscache.CacheableLockManagerImpl">
  <properties>
    <property name="time-out" value="15m" />
    <property name="jbosscache-configuration" value="test-jbosscache-lock.xml" />
    <property name="jgroups-configuration" value="udp-mux.xml" />
    <property name="jgroups-multiplexer-stack" value="true" />
    <property name="jbosscache-cluster-name" value="JCR-cluster-locks-ws" />
    <property name="jbosscache-cl-cache.jdbc.table.name" value="jcrlocks_ws" />
    <property name="jbosscache-cl-cache.jdbc.table.create" value="true" />
    <property name="jbosscache-cl-cache.jdbc.table.drop" value="false" />
    <property name="jbosscache-cl-cache.jdbc.table.primarykey" value="jcrlocks_ws_pk" />
    <property name="jbosscache-cl-cache.jdbc.fqn.column" value="fqn" />
    <property name="jbosscache-cl-cache.jdbc.fqn.type" value="AUTO"/>
    <property name="jbosscache-cl-cache.jdbc.node.column" value="node" />
    <property name="jbosscache-cl-cache.jdbc.node.type" value="AUTO"/>
    <property name="jbosscache-cl-cache.jdbc.parent.column" value="parent" />
    <property name="jbosscache-cl-cache.jdbc.datasource" value="jdbcjcr" />
  </properties>
</lock-manager>
```

Configuration requirements:

- `jbosscache-cl-cache.jdbc.fqn.column` and `jbosscache-cl-cache.jdbc.node.type` is the same as cache.jdbc.fqn.type and cache.jdbc.node.type in JBoss-Cache configuration. You can set those data types according to database type (See *Data Types in Different Databases*) or set it as AUTO (or do not set at all) and data type will be detected automatically.

- As you see, jgroups-configuration is moved to separate the configuration file - udp-mux.xml. In this case, the udp-mux.xml file is a common JGroup configuration for all components (QueryHandler, Cache, LockManager), but we can still create our own configuration.

`our-udp-mux.xml`

```
<protocol_stacks>
  <stack name="jcr.stack">
    <config>
                              <UDP    mcast_addr="228.10.10.10"    mcast_port="45588"    tos="8"
ucast_recv_buf_size="20000000"
                                ucast_send_buf_size="640000"    mcast_recv_buf_size="25000000"
mcast_send_buf_size="640000" loopback="false"
                                  discard_incompatible_packets="true"    max_bundle_size="64000"
max_bundle_timeout="30"
                            use_incoming_packet_handler="true"   ip_ttl="2"   enable_bundling="true"
enable_diagnostics="true"
             thread_naming_pattern="cl" use_concurrent_stack="true" thread_pool.enabled="true"
thread_pool.min_threads="2"
                                thread_pool.max_threads="8"    thread_pool.keep_alive_time="5000"
thread_pool.queue_enabled="true"
                          thread_pool.queue_max_size="1000"   thread_pool.rejection_policy="discard"
oob_thread_pool.enabled="true"
                              oob_thread_pool.min_threads="1"    oob_thread_pool.max_threads="8"
oob_thread_pool.keep_alive_time="5000"
                 oob_thread_pool.queue_enabled="false"  oob_thread_pool.queue_max_size="100"
oob_thread_pool.rejection_policy="Run" />

      <PING timeout="2000" num_initial_members="3" />
      <MERGE2 max_interval="30000" min_interval="10000" />
      <FD_SOCK />
      <FD timeout="10000" max_tries="5" shun="true" />
      <VERIFY_SUSPECT timeout="1500" />
      <BARRIER />
            <pbcast.NAKACK  use_stats_for_retransmission="false"  exponential_backoff="150"
use_mcast_xmit="true"
        gc_lag="0" retransmit_timeout="50,300,600,1200" discard_delivered_msgs="true" />
      <UNICAST timeout="300,600,1200" />
                              <pbcast.STABLE    stability_delay="1000"    desired_avg_gossip="50000"
max_bytes="1000000" />
      <VIEW_SYNC avg_send_interval="60000" />
    <pbcast.GMS print_local_addr="true" join_timeout="3000" shun="false" view_bundling="true"
/>
      <FC max_credits="500000" min_threshold="0.20" />
```

```
        <FRAG2 frag_size="60000" />
        <!--pbcast.STREAMING_STATE_TRANSFER /-->
        <pbcast.STATE_TRANSFER />
        <!-- pbcast.FLUSH  /-->
      </config>
   </stack>
</protocol_stacks>
```

Data Types in Different Databases

Table 23.1. FQN type and node type in different databases

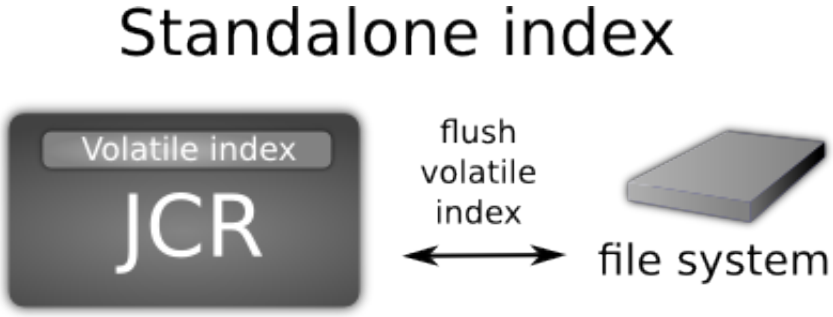
DataBase name	Node data type	FQN data type
default	BLOB	VARCHAR(512)
HSSQL	OBJECT	VARCHAR(512)
MySQL	LONGBLOB	VARCHAR(512)
ORACLE	BLOB	VARCHAR2(512)
PostgreSQL	bytea	VARCHAR(512)
MSSQL	VARBINARY(MAX)	VARCHAR(512)
DB2	BLOB	VARCHAR(512)
Sybase	IMAGE	VARCHAR(512)
Ingres	long byte	VARCHAR(512)

QueryHandler configuration

Indexing in clustered environment

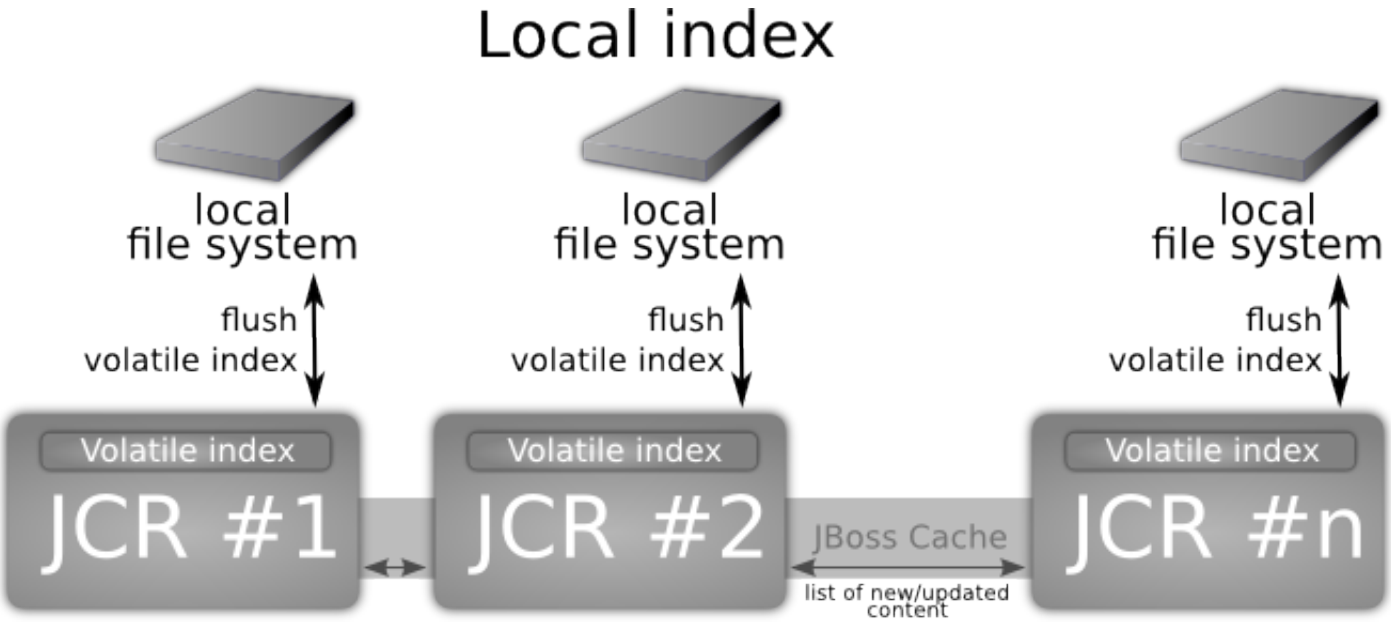
JCR offers multiple indexing strategies. They include both for standalone and clustered environments using the advantages of running in a single JVM or doing the best to use all resources available in cluster. JCR uses Lucene library as underlying search and indexing engine, but it has several limitations that greatly reduce possibilities and limits the usage of cluster advantages. That's why eXo JCR offers three strategies that are suitable for it's own usecases. They are standalone, clustered with shared index and clustered with local indexes. Each one has it's pros and cons.

Stanadlone strategy provides a stack of indexes to achieve greater performance within single JVM.



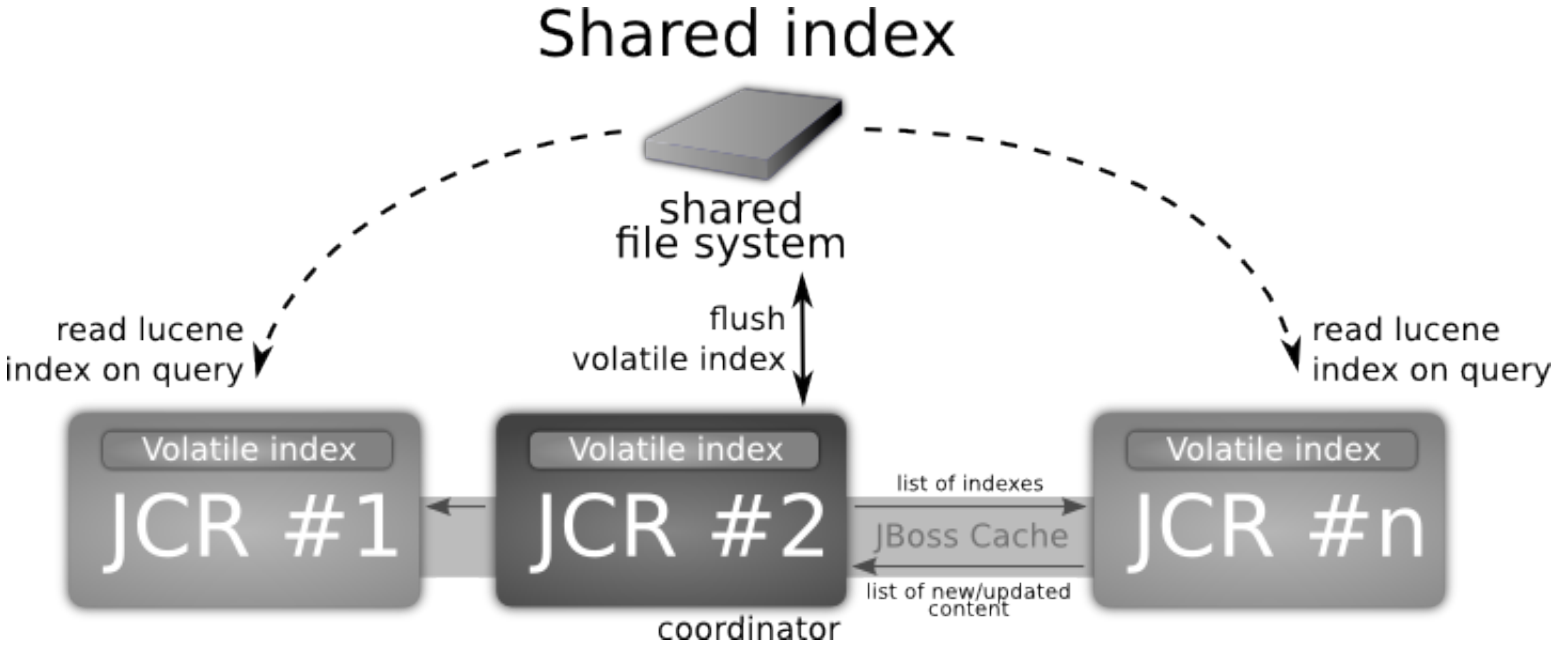
It combines in-memory buffer index directory with delayed file-system flushing. This index is called "Volatile" and it is invoked in searches also. Within some conditions volatile index is flushed to the persistent storage (file system) as new index directory. This allows to achieve great results for write operations.

Clustered implementation with local indexes is built upon same strategy with volatile in-memory index buffer along with delayed flushing on persistent storage.



As this implementation designed for clustered environment it has additional mechanisms for data delivery within cluster. Actual text extraction jobs done on the same node that does content operations (i.e. write operation). Prepared "documents" (Lucene term that means block of data ready for indexing) are replicated withing cluster nodes and processed by local indexes. So each cluster instance has the same index content. When new node joins the cluster it has no initial index, so it must be created. There are some supported ways of doing this operation. The simplest is to simply copy the index manually but this is not intended for use. If no initial index found JCR uses automated sceneries. They are controlled via configuration (see "index-recovery-mode" parameter) offering full re-indexing from database or copying from another cluster node.

For some reasons having a multiple index copies on each instance can be costly. So shared index can be used instead (see diagram below).



This indexing strategy combines advantages of in-memory index along with shared persistent index offering "near" real time search capabilities. This means that newly added content is accessible via search practically immediately. This strategy allows nodes to index data in their own volatile (in-memory) indexes, but persistent indexes are managed by single "coordinator" node only. Each cluster instance has a read access for shared index to perform queries combining search results found in own in-memory index also. Take in account that shared folder must be configured in your system environment (i.e. mounted NFS folder). But this strategy in some extremely rare cases can have a bit different volatile indexes within cluster instances for a while. In a few seconds they will be up2date.

See more about *Search Configuration*.

Configuration

Query-handler configuration overview

Configuration example:

```
<workspace name="ws">
  <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
    <properties>
      <property name="index-dir" value="shareddir/index/db1/ws" />
      <property name="changesfilter-class"


 />
      <property name="jbosscache-configuration" value="jbosscache-indexer.xml" />
      <property name="jgroups-configuration" value="udp-mux.xml" />
      <property name="jgroups-multiplexer-stack" value="true" />
      <property name="jbosscache-cluster-name" value="JCR-cluster-indexer-ws" />
      <property name="max-volatile-time" value="60" />
      <property name="rdbms-reindexing" value="true" />
      <property name="reindexing-page-size" value="1000" />
      <property name="index-recovery-mode" value="from-coordinator" />
                                                <property        name="index-recovery-filter"
 value="org.exoplatform.services.jcr.impl.core.query.lucene.DocNumberRecoveryFilter" />
    </properties>
  </query-handler>
</workspace>
```

Table 24.1. Config properties description

Property name	Description
index-dir	path to index

Property name	Description
changesfilter-class	template of JBoss-cache configuration for all query-handlers in repository
jbosscache-configuration	template of JBoss-cache configuration for all query-handlers in repository
jgroups-configuration	jgroups-configuration is template configuration for all components (search, cache, locks) [Add link to document describing template configurations]
jgroups-multiplexer-stack	[TODO about jgroups-multiplexer-stack - add link to JBoss doc]
jbosscache-cluster-name	cluster name (must be unique)
max-volatile-time	max time to live for Volatile Index
rdbms-reindexing	indicate that need to use rdbms reindexing mechanism if possible, the default value is true
reindexing-page-size	maximum amount of nodes which can be retrieved from storage for re-indexing purpose, the default value is 100
index-recovery-mode	If the parameter has been set to **from-indexing**, so a full indexing will be automatically launched (default behavior), if the parameter has been set to **from-coordinator**, the index will be retrieved from coordinator
index-recovery-filter	Defines implementation class or classes of RecoveryFilters, the mechanism of index synchronization for Local Index strategy.
async-reindexing	Controls the process of re-indexing on JCR's startup. If flag set, indexing will be launched asynchronously, without blocking the JCR. Default is "false".

Note

If you use postgreSQL and the parameter rdbms-reindexing is set to true, the performances of the queries used while indexing can be improved by setting the parameter "enable_seqscan" to "off" or "default_statistics_target" to at least "50" in the configuration of your database. Then you need to restart DB server and make analyze of the JCR_SVALUE (or JCR_MVALUE) table.

Note

If you use DB2 and the parameter rdbms-reindexing is set to true, the performance of the queiries used while indexing can be improved by making statisticks on tables by running "RUNSTATS ON TABLE <scheme>.<table> WITH DISTRIBUTION AND INDEXES ALL" for JCR_SITEM (or JCR_MITEM) and JCR_SVALUE (or JCR_MVALUE) tables.

Standalone strategy

When running JCR in standalone usually standalone indexing is used also. Such parameters as "changesfilter-class", "jgroups-configuration" and all the "jbosscache-*" must be skipped and not defined. Like the configuration below.

```
<workspace name="ws">
  <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
    <properties>
      <property name="index-dir" value="shareddir/index/db1/ws" />
      <property name="max-volatile-time" value="60" />
      <property name="rdbms-reindexing" value="true" />
      <property name="reindexing-page-size" value="1000" />
      <property name="index-recovery-mode" value="from-coordinator" />
    </properties>
  </query-handler>
</workspace>
```

Cluster-ready indexing strategies

For both cluster-ready implementations JBoss Cache, JGroups and Changes Filter values must be defined. Shared index requires some kind of remote or shared file system to be attached in a system (i.e. NFS, SMB or etc). Indexing directory ("indexDir" value) must point to it. Setting "changesfilter-class" to "org.exoplatform.services.jcr.impl.core.query.jbosscache.JBossCacheIndexChangesFilter" will enable shared index implementation.

```
<workspace name="ws">
  <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
    <properties>
      <property name="index-dir" value="/mnt/nfs_drive/index/db1/ws" />
      <property name="changesfilter-class"


 />
      <property name="jbosscache-configuration" value="jbosscache-indexer.xml" />
```

```
        <property name="jgroups-configuration" value="udp-mux.xml" />
        <property name="jgroups-multiplexer-stack" value="true" />
        <property name="jbosscache-cluster-name" value="JCR-cluster-indexer-ws" />
        <property name="max-volatile-time" value="60" />
        <property name="rdbms-reindexing" value="true" />
        <property name="reindexing-page-size" value="1000" />
        <property name="index-recovery-mode" value="from-coordinator" />
      </properties>
    </query-handler>
</workspace>
```

In order to use cluster-ready strategy based on local indexes, when each node has own copy of index on local file system, the following configuration must be applied. Indexing directory must point to any folder on local file system and "changesfilter-class" must be set to "org.exoplatform.services.jcr.impl.core.query.jbosscache.LocalIndexChangesFilter".

```
<workspace name="ws">
  <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
    <properties>
      <property name="index-dir" value="/mnt/nfs_drive/index/db1/ws" />
      <property name="changesfilter-class"
       value="org.exoplatform.services.jcr.impl.core.query.jbosscache.LocalIndexChangesFilter"
 />
      <property name="jbosscache-configuration" value="jbosscache-indexer.xml" />
      <property name="jgroups-configuration" value="udp-mux.xml" />
      <property name="jgroups-multiplexer-stack" value="true" />
      <property name="jbosscache-cluster-name" value="JCR-cluster-indexer-ws" />
      <property name="max-volatile-time" value="60" />
      <property name="rdbms-reindexing" value="true" />
      <property name="reindexing-page-size" value="1000" />
      <property name="index-recovery-mode" value="from-coordinator" />
    </properties>
  </query-handler>
</workspace>
```

Local Index Recovery Filters

Common usecase for all cluster-ready applications is a hot joining and leaving of processing units. Node that is joining cluster for the first time or node joining after some downtime, they all must be in a synchronized state. When having a deal with shared value storages, databases and indexes, cluster nodes are synchronized anytime. But it's an issue when local index strategy used. If new node joins cluster, having no index it is retrieved or recreated. Node can be restarted also and thus

index not empty. By default existing index is thought to be actual, but can be outdated. JCR offers a mechanism called RecoveryFilters that will automatically retrieve index for the joining node on startup. This feature is a set of filters that can be defined via QueryHandler configuration:

```
<property name="index-recovery-filter"
 value="org.exoplatform.services.jcr.impl.core.query.lucene.DocNumberRecoveryFilter" />
```

Filter number is not limited so they can be combined:

```
<property name="index-recovery-filter"
 value="org.exoplatform.services.jcr.impl.core.query.lucene.DocNumberRecoveryFilter" />
<property name="index-recovery-filter"
 value="org.exoplatform.services.jcr.impl.core.query.lucene.SystemPropertyRecoveryFilter" />
```

If any one returns fires, the index is re-synchronized. This feature uses standard index recovery mode defined by previously described parameter (can be "from-indexing" (default) or "from-coordinator")

```
<property name="index-recovery-mode" value="from-coordinator" />
```

There are couple implementations of filters:

- org.exoplatform.services.jcr.impl.core.query.lucene.DummyRecoveryFilter: always returns true, for cases when index must be force resynchronized (recovered) each time;
- org.exoplatform.services.jcr.impl.core.query.lucene.SystemPropertyRecoveryFilter : return value of system property "org.exoplatform.jcr.recoveryfilter.forcereindexing". So index recovery can be controlled from the top without changing documentation using system properties;
- org.exoplatform.services.jcr.impl.core.query.lucene.ConfigurationPropertyRecoveryFilter : return value of QueryHandler configuration property "index-recovery-filter-forcereindexing". So index recovery can be controlled from configuration separately for each workspace. I.e:

```
<property name="index-recovery-filter"

 />
<property name="index-recovery-filter-forcereindexing" value="true" />
```

- org.exoplatform.services.jcr.impl.core.query.lucene.DocNumberRecoveryFilter : checks number of documents in index on coordinator side and self-side. Return true if differs. Advantage

of this filter comparing to other, it will skip reindexing for workspaces where index wasn't modified. I.e. there is 10 repositories with 3 workspaces in each one. Only one is really heavily used in cluster : frontend/production. So using this filter will only reindex those workspaces that are really changed, without affecting other indexes thus greatly reducing startup time.

JBoss-Cache template configuration

JBoss-Cache template configuration for query handler is about the same for both clustered strategies.

jbosscache-indexer.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<jbosscache                                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="urn:jboss:jbosscache-core:config:3.1">

                          <locking          useLockStriping="false"          concurrencyLevel="50000"
 lockParentForChildInsertRemove="false"
     lockAcquisitionTimeout="20000" />
  <!-- Configure the TransactionManager -->
                                                                                          <transaction

 />

  <clustering mode="replication" clusterName="${jbosscache-cluster-name}">
     <stateRetrieval timeout="20000" fetchInMemoryState="false" />
     <jgroupsConfig multiplexerStack="jcr.stack" />
     <sync />
  </clustering>
  <!-- Eviction configuration -->
  <eviction wakeUpInterval="5000">
                                  <default       algorithmClass="org.jboss.cache.eviction.FIFOAlgorithm"
 eventQueueSize="1000000">
        <property name="maxNodes" value="10000" />
        <property name="minTimeToLive" value="60000" />
     </default>
  </eviction>

</jbosscache>
```

See more about template configurations *here*.

Asynchronous reindexing

Managing a big set of data using JCR in production environment sometimes requires special operations with Indexes, stored on File System. One of those maintenance operations is a recreation of it. Also called "re-indexing". There are various usecases when it's important to do. They include hardware faults, hard restarts, data-corruption, migrations and JCR updates that brings new features related to index. Usually index re-creation requested on server's startup or in runtime.

On startup indexing

Common usecase for updating and re-creating the index is to stop the server and manually remove indexes for workspaces requiring it. When server will be started, missing indexes are automatically recovered by re-indexing. JCR Supports direct RDBMS re-indexing, that usually is faster than ordinary and can be configured via QueryHandler parameter "rdbms-reindexing" set to "true" (for more information please refer to "Query-handler configuration overview"). New feature to introduce is asynchronous indexing on startup. Usually startup is blocked until process is finished. Block can take any period of time, depending on amount of data persisted in repositories. But this can be resolved by using an asynchronous approaches of startup indexation. Saying briefly, it performs all operations with index in background, without blocking the repository. This is controlled by the value of "async-reindexing" parameter in QueryHandler configuration. With asynchronous indexation active, JCR starts with no active indexes present. Queries on JCR still can be executed without exceptions, but no results will be returned until index creation completed. Checking index state is possible via QueryManagerImpl:

```
boolean online =
((QueryManagerImpl)Workspace.getQueryManager()).getQueryHandeler().isOnline();
```

"OFFLINE" state means that index is currently re-creating. When state changed, corresponding log event is printed. From the start of background task index is switched to "OFFLINE", with following log event :

```
[INFO] Setting index OFFLINE (repository/production[system]).
```

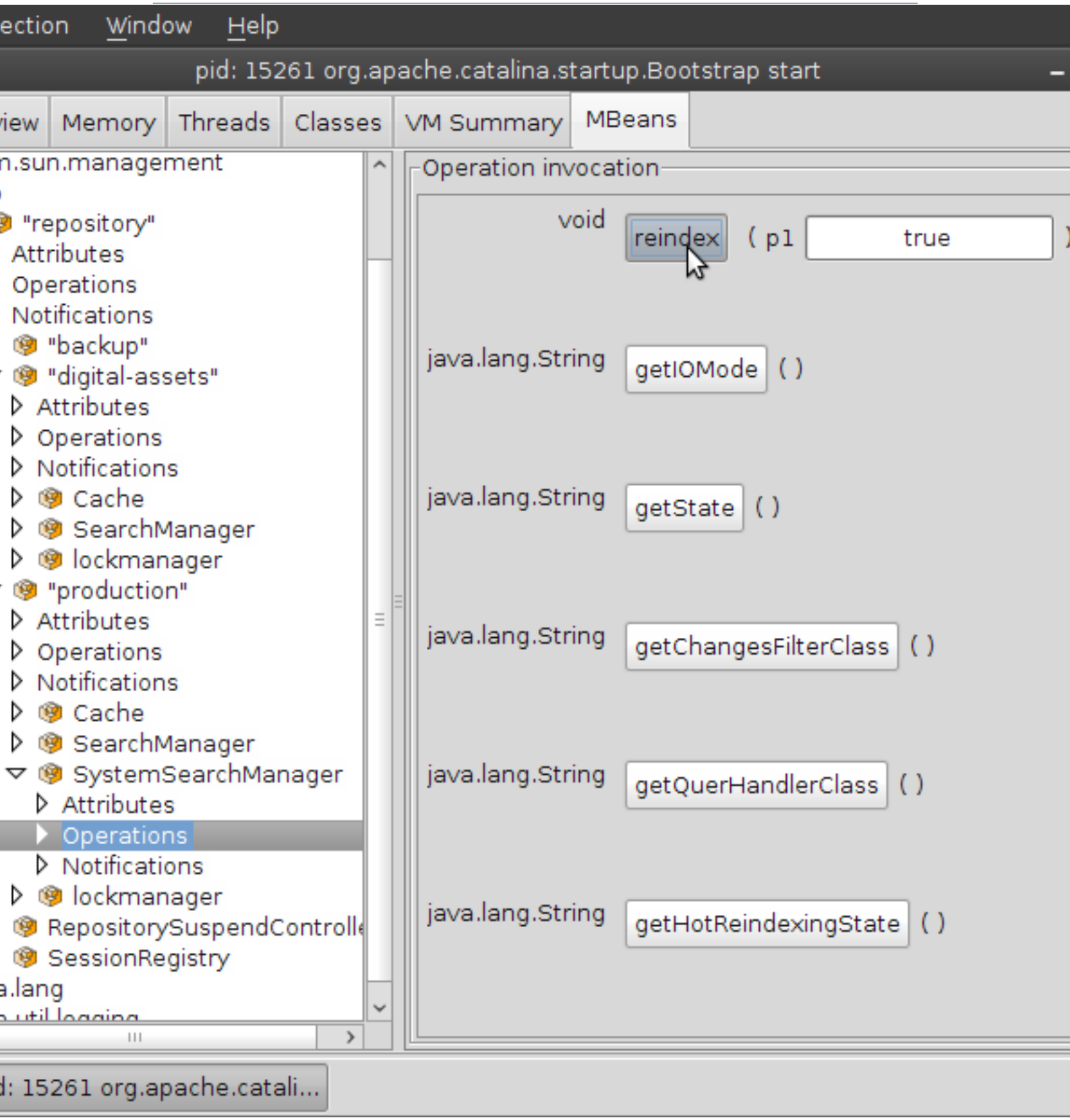
When process finished, two events are logged :

```
[INFO] Created initial index for 143018 nodes (repository/production[system]).
[INFO] Setting index ONLINE (repository/production[system]).
```

Those two log lines indicates the end of process for workspace given in brackets. Calling isOnline() as mentioned above, will also return true.

Hot Asynchronous Workspace Reindexing via JMX

Some hard system faults, error during upgrades, migration issues and some other factors may corrupt the index. Most likely end customers would like the production systems to fix index issues in run-time, without delays and restarts. Current versions of JCR supports "Hot Asynchronous Workspace Reindexing" feature. It allows end-user (Service Administrator) to launch the process in background without stopping or blocking whole application by using any JMX-compatible console (see screenshot below, "JConsole in action").



Server can continue working as expected while index is recreated. This depends on the flag "allow queries", passed via JMX interface to reindex operation invocation. If the flag set, then application continues working. But there is one critical limitation the end-users must be aware. Index is frozen

while background task is running. It meant that queries are performed on index present on the moment of task startup and data written into repository after startup won't be available through the search until process finished. Data added during re-indexation is also indexed, but will be available only when task is done. Briefly, JCR makes the snapshot of indexes on asynch task startup and uses it for searches. When operation finished, stale indexes replaced by newly created including newly added data. If flag "allow queries" is set to false, then all queries will throw an exception while task is running. Current state can be acquired using the following JMX operation:

- getHotReindexingState() - returns information about latest invocation: start time, if in progress or finish time if done.

Notices

First of all, can't launch Hot re-indexing via JMX if index is already in offline mode. It means that index is currently is invoked in some operations, like re-indexing at startup, copying in cluster to another node or whatever. Another important this is Hot Asynchronous Reindexing via JMX and "on startup" reindexing are completely different features. So you can't get the state of startup reindexing using command getHotReindexingState in JMX interface, but there are some common JMX operations:

- getIOMode - returns current index IO mode (READ_ONLY / READ_WRITE), belongs to clustered configuration states;
- getState - returns current state: ONLINE / OFFLINE.

Advanced tuning

Lucene tuning

As mentioned above, JCR Indexing is based on Lucene indexing library as underlying search engine. It uses Directories to store index and manages access to index by Lock Factories. By default JCR implementation uses optimal combination of Directory implementation and Lock Factory implementation. When running on OS different from Windows, NIOFSDirectory implementation used. And SimpleFSDirectory for Windows stations. NativeFSLockFactory is an optimal solution for wide variety of cases including clustered environment with NFS shared resources. But those default can be overridden with the help of system properties. There are two properties: "org.exoplatform.jcr.lucene.store.FSDirectoryLockFactoryClass" and "org.exoplatform.jcr.lucene.FSDirectory.class" that are responsible for changing default behavior. First one defines implementation of abstract Lucene LockFactory class and the second one sets implementation class for FSDirectory instances. For more information please refer to Lucene documentation. But be sure You know what You are changing. JCR allows end users to change implementation classes of Lucene internals, but doesn't guarantee it's stability and functionality.

JBossTransactionsService

Introduction

JBossTransactionsService implements eXo *TransactionService* and provides access to *JBoss Transaction Service (JBossTS)* [http://www.jboss.org/jbosstm/] JTA implementation via eXo container dependency.

TransactionService used in JCR cache *org.exoplatform.services.jcr.impl.dataflow.persistent.jbosscache.JBossCacheWorkspaceStorageCache* implementaion. See *Cluster configuration* for example.

Configuration

Example configuration:

```
<component>
  <key>org.exoplatform.services.transaction.TransactionService</key>
  <type>org.exoplatform.services.transaction.jbosscache.JBossTransactionsService</type>
  <init-params>
    <value-param>
      <name>timeout</name>
      <value>3000</value>
    </value-param>
  </init-params>
</component>
```

timeout - XA transaction timeout in seconds

TransactionManagerLookup

Configuration

It's JBossCache class registered as eXo container component in configuration.xml file.

```
<component>
   <key>org.jboss.cache.transaction.TransactionManagerLookup</key>
   <type>org.jboss.cache.transaction.JBossStandaloneJTAManagerLookup</type>
</component>
```

JBossStandaloneJTAManagerLookup used in standalone environment. Bur for Application Server environment use GenericTransactionManagerLookup.

RepositoryCreationService

Intro

RepositoryCreationService is the service for creation repositories in runtime. The service can be used in standalone or cluster environment.

Dependencies

RepositoryConfigurationService depends to next components:

- *DBCreator* - DBCreator used to create new database for each unbinded datasource.
- *BackupManager* - BackupManager used to created repository from backup.
- *RPCService* - RPCService used for communication between cluster-nodes

> **Note**
>
> RPCService may not be configured - in this case, RepositoryService will work as standalone service.

How it works

- User executes reserveRepositoryName(String repositoryName) - client-node calls coordinator-node to reserve repositoryName. If this name is already reserved or repository with this name exist, client-node will fetch RepositoryCreationException. If not Client will get token string.
- than user executes createRepository(String backupId, RepositoryEntry rEntry, String token). Coordinator-node checks the token, and creates Repository.
- whan repository become created - user-node broadcast message to all clusterNodes with RepositoryEntry, so each cluster node starts new Repository.

There is two ways to create repositry: make it in single step - just call createRepository(String backupId, RepositoryEntry); or reserve repositoryName at first (reserveRepositoryName(String repositoryName)), than create reserved repository (createRepository(String backupId, RepositoryEntry rEntry, String token)).

Configuration

RepositoryCreationService configuration

```
<component>
  <key>org.exoplatform.services.jcr.ext.backup.BackupManager</key>
  <type>org.exoplatform.services.jcr.ext.backup.impl.BackupManagerImpl</type>
```

```
  <init-params>
    <properties-param>
      <name>backup-properties</name>
      <property name="backup-dir" value="target/backup" />
    </properties-param>
  </init-params>
</component>

<component>
  <key>org.exoplatform.services.database.creator.DBCreator</key>
  <type>org.exoplatform.services.database.creator.DBCreator</type>
  <init-params>
    <properties-param>
      <name>db-connection</name>
      <description>database connection properties</description>
      <property name="driverClassName" value="org.hsqldb.jdbcDriver" />
      <property name="url" value="jdbc:hsqldb:file:target/temp/data/" />
      <property name="username" value="sa" />
      <property name="password" value="" />
    </properties-param>
    <properties-param>
      <name>db-creation</name>
      <description>database creation properties</description>
      <property name="scriptPath" value="src/test/resources/test.sql" />
      <property name="username" value="sa" />
      <property name="password" value="" />
    </properties-param>
  </init-params>
</component>

<component>
  <key>org.exoplatform.services.rpc.RPCService</key>
  <type>org.exoplatform.services.rpc.impl.RPCServiceImpl</type>
  <init-params>
    <value-param>
      <name>jgroups-configuration</name>
      <value>jar:/conf/standalone/udp-mux.xml</value>
    </value-param>
    <value-param>
      <name>jgroups-cluster-name</name>
      <value>RPCService-Cluster</value>
    </value-param>
    <value-param>
      <name>jgroups-default-timeout</name>
```

```
        <value>0</value>
      </value-param>
    </init-params>
</component>

<component>
   <key>org.exoplatform.services.jcr.ext.repository.creation.RepositoryCreationService</key>
   <type>
      org.exoplatform.services.jcr.ext.repository.creation.RepositoryCreationServiceImpl
   </type>
     <init-params>
        <value-param>
          <name>factory-class-name</name>
          <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
        </value-param>
     </init-params>
</component>
```

- factory-class-name - is not mandatory parameter, indicates what the factory need to use to create DataSource objects

RepositoryCreationService Interface

```
public interface RepositoryCreationService
{
   /**
    * Reserves, validates and creates repository in a simplified form.
    *
    * @param rEntry - repository Entry - note that datasource must not exist.
    * @param backupId - backup id
    * @param creationProps - storage creation properties
    * @throws RepositoryConfigurationException
     *          if some exception occurred during repository creation or repository name is absent
 in reserved list
    * @throws RepositoryCreationServiceException
     *          if some exception occurred during repository creation or repository name is absent
 in reserved list
    */
     void createRepository(String backupId, RepositoryEntry rEntry, StorageCreationProperties
 creationProps)
      throws RepositoryConfigurationException, RepositoryCreationException;n

   /**
```

```
   * Reserves, validates and creates repository in a simplified form.
   *
   * @param rEntry - repository Entry - note that datasource must not exist.
   * @param backupId - backup id
   * @throws RepositoryConfigurationException
    *          if some exception occurred during repository creation or repository name is absent
in reserved list
   * @throws RepositoryCreationServiceException
    *          if some exception occurred during repository creation or repository name is absent
in reserved list
   */
             void createRepository(String backupId, RepositoryEntry rEntry) throws
RepositoryConfigurationException,
     RepositoryCreationException;

  /**
    * Reserve repository name to prevent repository creation with same name from other place
in same time
   * via this service.
   *
   * @param repositoryName - repositoryName
   * @return repository token. Anyone obtaining a token can later create a repository of reserved
name.
   * @throws RepositoryCreationServiceException if can't reserve name
   */
  String reserveRepositoryName(String repositoryName) throws RepositoryCreationException;

  /**
   * Creates repository, using token of already reserved repository name.
    * Good for cases, when repository creation should be delayed or made asynchronously in
dedicated thread.
   *
   * @param rEntry - repository entry - note, that datasource must not exist
   * @param backupId - backup id
   * @param rToken - token
   * @param creationProps - storage creation properties
   * @throws RepositoryConfigurationException
    *          if some exception occurred during repository creation or repository name is absent
in reserved list
   * @throws RepositoryCreationServiceException
    *          if some exception occurred during repository creation or repository name is absent
in reserved list
   */
```

```
        void createRepository(String backupId, RepositoryEntry rEntry, String rToken,
StorageCreationProperties creationProps)
    throws RepositoryConfigurationException, RepositoryCreationException;

  /**
   * Creates  repository, using token of already reserved repository name. Good for cases, when
repository creation should be delayed or
   * made asynchronously in dedicated thread.
   *
   * @param rEntry - repository entry - note, that datasource must not exist
   * @param backupId - backup id
   * @param rToken - token
   * @throws RepositoryConfigurationException
    *          if some exception occurred during repository creation or repository name is absent
in reserved list
   * @throws RepositoryCreationServiceException
    *          if some exception occurred during repository creation or repository name is absent
in reserved list
   */
  void createRepository(String backupId, RepositoryEntry rEntry, String rToken)
    throws RepositoryConfigurationException, RepositoryCreationException;

  /**
   * Remove previously created repository.
   *
   * @param repositoryName - the repository name to delete
   * @param forceRemove - force close all opened sessions
   * @throws RepositoryCreationServiceException
   *          if some exception occurred during repository removing occurred
   */
        void removeRepository(String repositoryName, boolean forceRemove) throws
RepositoryCreationException;

}
```

Conclusions and restrictions

- Each datasource in RepositoryEntry of new Repository must have unbinded datasources. Thats mean, such datasource must have not databases behind them. This restriction exists to avoid corruption of existing repositories data.
- RPCService is optional component, but without it, RepositoryCreatorService can not communicate with other cluster-nodes and works as standalone.

JCR Query Usecases

Intro

JCR supports two query languages - JCR and XPath. A query, whether XPath or SQL, specifies a subset of nodes within a workspace, called the result set. The result set constitutes all the nodes in the workspace that meet the constraints stated in the query.

Query Lifecycle

Query Creation and Execution

SQL

```
// get QueryManager
QueryManager queryManager = workspace.getQueryManager();
// make SQL query
Query query = queryManager.createQuery("SELECT * FROM nt:base ", Query.SQL);
// execute query
QueryResult result = query.execute();
```

XPath

```
// get QueryManager
QueryManager queryManager = workspace.getQueryManager();
// make XPath query
Query query = queryManager.createQuery("//element(*,nt:base)", Query.XPATH);
// execute query
QueryResult result = query.execute();
```

Query Result Processing

```
// fetch query result
QueryResult result = query.execute();
```

Now we can get result in an iterator of nodes:

```
NodeIterator it = result.getNodes();
```

or we get the result in a table:

```
// get column names
String[] columnNames = result.getColumnNames();
// get column rows
RowIterator rowIterator = result.getRows();
while(rowIterator.hasNext()){
  // get next row
  Row row = rowIterator.nextRow();
  // get all values of row
  Value[] values = row.getValues();
}
```

Scoring

The result returns a score for each row in the result set. The score contains a value that indicates a rating of how well the result node matches the query. A high value means a better matching than a low value. This score can be used for ordering the result.

eXo JCR Scoring is a mapping of Lucene scoring. For a more in-depth understanding, please study *Lucene documentation* [http://lucene.apache.org/java/2_4_1/scoring.html].

jcr:score counted in next way - (lucene score)*1000f.

Score may be increased for specified nodes, see *Index Boost Value*

Also, see an example *Order by Score*

Query result settings

- *Set Offset And Limit*

Type Constraints

- *Find All Nodes*
- *Find Nodes by Primary Type*
- *Find Nodes by Mixin Type*

Property Constraints

- *Property Comparison*

- *LIKE Constraint*
- *Escaping in LIKE Statements*
- *NOT Constraint*
- *AND Constraint*
- *OR Constraint*
- *Property Existence Constraint*
- *Upper and Lower Case Constraints*
- *Date Property Comparison*
- *Node Name Constraint*
- *Multivalue Property Comparison*

Path Constraint

- *Exact Path Constraint*
- *Child Node Constraint*
- *Find All Descendant Nodes*

Ordering specifing

- *Order by Property*
- *Order by Descendant Node Property*
- *Order by Score*
- *Order by Path or Name*

Fulltext Search

- *Fulltext Search by Property*
- *Fulltext Search by All Properties*
- *Find nt:file document by content of child jcr:content node*

- *How to set new Analyzer. Accent symblos ignoring*

Indexing rules and additional features

- *Aggregation rule*
- *Search Result Highlighting*
- *Index Boost Value*
- *Exclusion from the Node Scope Index*
- *Regular expressions as property name in indexing rule*
- *Synonim Provider*
- *Spell Checking*
- *Find Similar Nodes*

Query Examples

SetOffset and SetLimit

Select all nodes with primary type 'nt:unstructured' and returns only 3 nodes starting with the second node in the list.

Common info

QueryImpl class has two methods: one to indicate how many results shall be returned at most, and another to fix the starting position.

- setOffset(long offset) - Sets the start offset of the result set.
- setLimit(long position) - Sets the maximum size of the result set.

Repository structure

Repository contains mix:title nodes, where jcr:title has different values.

- root
  - node1 (nt:unstructured)
  - node2 (nt:unstructured)

- node3 (nt:unstructured)
- node4 (nt:unstructured)
- node5 (nt:unstructured)
- node6 (nt:unstructured)

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM nt:unstructured";
QueryImpl query = (QueryImpl)queryManager.createQuery(sqlStatement, Query.SQL);
//return starting with second result
query.setOffset(1);
// return 3 results
query.setLimit(3);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

In usual case (without using setOffset and setLimit methods), Node iterator returns all nodes (node1...node6). But in our case NodeIterator will return "node2","node3" and "node4".

\[node1 **node2 node3 node4** node5 node6\]

Finding All Nodes

Find all nodes in the repository. Only those nodes are found to which the session has READ permission. See also *Access Control*.

Repository structure:

Repository contains many different nodes.

- root
  - folder1 (nt:folder)
    - document1 (nt:file)
    - folder2 (nt:folder)
      - document2 (nt:unstructured)
      - document3 (nt:folder)

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM nt:base";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,nt:base)";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Let's get nodes:

```
NodeIterator it = result.getNodes();
```

```
if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

NodeIterator will return "folder1", "folder2","document1","document2","document3", and each other nodes in workspace if they are here.

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
}
```

Table content is

Table 28.1. Table content

jcr:path	jcr:score
/folder1	1000
/folder1/document1	1000
/folder1/folder2	1000
/folder1/folder2/document2	1000
/folder1/folder2/document3	1000
...	...

Finding Nodes by Primary Type

Find all nodes whose primary type is "nt:file".

Repository structure:

The repository contains nodes with different primary types and mixin types.

- root

- document1 primarytype = "nt:unstructured" mixintype = "mix:title"
- document2 primarytype = "nt:file" mixintype = "mix:lockable"
- document3 primarytype = "nt:file" mixintype = "mix:title"

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM nt:file";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,nt:file)";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

NodeIterator will return "document2" and "document3".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
}
```

The table content is

Table 28.2. Table content

jcr:path	jcr:score
/document2	2674
/document3	2674

Finding Nodes by Mixin Type

Find all nodes in repository, that contain a mixin type "mix:title".

Repository structure:

The repository contains nodes with different primary types and mixin types.

- root
  - document1 primarytype = "nt:unstructured" mixintype = "mix:title"
  - document2 primarytype = "nt:file" mixintype = "mix:lockable"
  - document3 primarytype = "nt:file" mixintype = "mix:title"

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM mix:title";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,mix:title)";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

The NodeIterator will return "document1" and "document3".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
}
```

Table content is

Table 28.3. Table content

jcr:title	...	jcr:path	jcr:score
First document	...	/document1	2674
Second document	...	/document3	2674

Property Comparison

Find all nodes with mixin type 'mix:title' where the prop_pagecount property contains a value less than 90. Only select the title of each node.

Repository structure:

Repository contains several mix:title nodes, where each prop_pagecount contains a different value.

- root
  - document1 (mix:title) jcr:title="War and peace" prop_pagecount=1000
  - document2 (mix:title) jcr:title="Cinderella" prop_pagecount=100
  - document3 (mix:title) jcr:title="Puss in Boots" prop_pagecount=60

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT jcr:title FROM mix:title WHERE prop_pagecount < 90";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,mix:title)[@prop_pagecount < 90]/@jcr:title";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
  Node findedNode = it.nextNode();
}
```

The NodeIterator will return "document3".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
  Row row = rit.nextRow();
  // get values of the row
  Value[] values = row.getValues();
}
```

The table content is

Table 28.4. Table Content

jcr:title	jcr:path	jcr:score
Puss in Boots	/document3	1725

LIKE Constraint

Find all nodes with mixin type 'mix:title' and where the property 'jcr:title' starts with 'P'.

Note

See also the article about "*Find all mix:title nodes where jcr:title does NOT start with 'P*"

Repository structure:

The repository contains 3 mix:title nodes, where each jcr:title has a different value.

- root
  - document1 (mix:title) jcr:title="Star wars" jcr:description="Dart rules!!"
  - document2 (mix:title) jcr:title="Prison break" jcr:description="Run, Forest, run))"

- document3 (mix:title) jcr:title="Panopticum" jcr:description="It's imagine film"

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM mix:title WHERE jcr:title LIKE 'P%'";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,mix:title)[jcr:like(@jcr:title, 'P%')]";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

The NodeIterator will return "document2" and "document3".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
```

```
RowIterator rit = result.getRows();
while (rit.hasNext())
{
  Row row = rit.nextRow();
  // get values of the row
  Value[] values = row.getValues();
}
```

The table content is

Table 28.5. Table content

jcr:title	jcr:description	jcr:path	jcr:score
Prison break	Run, Forest, run))	/document2	4713
Panopticum	It's imagine film	/document3	5150

Escaping in LIKE Statements

Find all nodes with a mixin type 'mix:title' and whose property 'jcr:title' starts with 'P%ri'.

As you see "P%rison break" contains the symbol '%'. This symbol is reserved for LIKE comparisons. So what can we do?

Within the LIKE pattern, literal instances of percent ("%") or underscore ("_") must be escaped. The SQL ESCAPE clause allows the definition of an arbitrary escape character within the context of a single LIKE statement. The following example defines the backslash ' \' as escape character:

```
SELECT * FROM mytype WHERE a LIKE 'foo\%' ESCAPE '\'
```

XPath does not have any specification for defining escape symbols, so we must use the default escape character (' \').

Repository structure

The repository contains mix:title nodes, where jcr:title can have different values.

- root
  - document1 (mix:title) jcr:title="Star wars" jcr:description="Dart rules!!"
  - document2 (mix:title) jcr:title="P%rison break" jcr:description="Run, Forest, run))"
  - document3 (mix:title) jcr:title="Panopticum" jcr:description="It's imagine film"

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM mix:title WHERE jcr:title LIKE 'P#%ri%' ESCAPE '#'";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,mix:title)[jcr:like(@jcr:title, 'P\\%ri%')]";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

NodeIterator will return "document2".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
```

```
{
  Row row = rit.nextRow();
  // get values of the row
  Value[] values = row.getValues();
}
```

The table content is

Table 28.6. Table content

jcr:title	jcr:description	jcr:path	jcr:score
P%rison break	Run, Forest, run))	/document2	7452

NOT Constraint

Find all nodes with a mixin type 'mix:title' and where the property 'jcr:title' does NOT start with a 'P' symbol

Repository Structure

The repository contains a mix:title nodes, where the jcr:title has different values.

- root
  - document1 (mix:title) jcr:title="Star wars" jcr:description="Dart rules!!"
  - document2 (mix:title) jcr:title="Prison break" jcr:description="Run, Forest, run))"
  - document3 (mix:title) jcr:title="Panopticum" jcr:description="It's imagine film"

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM mix:title WHERE NOT jcr:title LIKE 'P%'";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,mix:title)[not(jcr:like(@jcr:title, 'P%'))]";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching the result

Let's get the nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

NodeIterator will return "document1".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
}
```

Table content is

Table 28.7. Table content

jcr:title	jcr:description	jcr:path	jcr:score
Star wars	Dart rules!!	/document1	4713

AND Constraint

Find all fairytales with a page count more than 90 pages.

How does it sound in jcr terms - Find all nodes with mixin type 'mix:title' where the property 'jcr:description' equals "fairytale" and whose "prop_pagecount" property value is less than 90.

Note

See also *Multivalue Property Comparison*.

Repository Structure:

The repository contains mix:title nodes, where prop_pagecount has different values.

- root
  - document1 (mix:title) jcr:title="War and peace" jcr:description="novel" prop_pagecount=1000
  - document2 (mix:title) jcr:title="Cinderella" jcr:description="fairytale" prop_pagecount=100
  - document3 (mix:title) jcr:title="Puss in Boots" jcr:description="fairytale" prop_pagecount=60

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM mix:title WHERE jcr:description = 'fairytale' AND
 prop_pagecount > 90";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,mix:title)[@jcr:description='fairytale' and @prop_pagecount
 > 90]";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching the Result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

NodeIterator will return "document2".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
}
```

Table content is:

Table 28.8. Table content

jcr:title	jcr:description	prop_pagecount	jcr:path	jcr:score
Cinderella	fairytale	100	/document2	7086

OR Constraint

Find all documents whose title is 'Cinderella' or whose description is 'novel'.

How does it sound in jcr terms? - Find all nodes with a mixin type 'mix:title' whose property 'jcr:title' equals "Cinderella" or whose "jcr:description" property value is "novel".

Repository Structure

The repository contains mix:title nodes, where jcr:title and jcr:description have different values.

- root
  - document1 (mix:title) jcr:title="War and peace" jcr:description="novel"

- document2 (mix:title) jcr:title="Cinderella" jcr:description="fairytale"
- document3 (mix:title) jcr:title="Puss in Boots" jcr:description="fairytale"

Query Execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM mix:title WHERE jcr:title = 'Cinderella' OR jcr:description
 = 'novel'";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,mix:title)[@jcr:title='Cinderella' or @jcr:description =
 'novel']";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching the Result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

NodeIterator will return "document1" and "document2".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
}
```

Table content is:

Table 28.9. Table content

jcr:title	jcr:description	jcr:path	jcr:score
War and peace	novel	/document1	3806
Cinderella	fairytale	/document2	3806

Property Existence Constraint

Find all nodes with a mixin type 'mix:title' where the property 'jcr:description' does not exist (is null).

Repository Structure

The repository contains mix:title nodes, in one of these nodes the jcr:description property is null.

- root
  - document1 (mix:title) jcr:title="Star wars" jcr:description="Dart rules!!"
  - document2 (mix:title) jcr:title="Prison break" jcr:description="Run, Forest, run))"
  - document3 (mix:title) jcr:title="Titanic" // The description property does not exist. This is the node we wish to find.

Query Execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
```

```
String sqlStatement = "SELECT * FROM mix:title WHERE jcr:description IS NULL";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = ""//element(*,mix:title)[not(@jcr:description)]"";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching the Result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

NodeIterator will return "document3".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
}
```

Table content is:

Table 28.10. Table content

jcr:title	jcr:description	jcr:path	jcr:score
Titanic	null	/document3	1947

Finding Nodes in a Case-Insensitive Way

Find all nodes with a mixin type 'mix:title' and where the property 'jcr:title' equals 'casesensitive' in lower or upper case.

Repository Structure

The repository contains mix:title nodes, whose jcr:title properties have different values.

- root
  - document1 (mix:title) jcr:title="CaseSensitive"
  - document2 (mix:title) jcr:title="casesensitive"
  - document3 (mix:title) jcr:title="caseSENSITIVE"

Query Execution

- UPPER case

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM mix:title WHERE UPPER(jcr:title) = 'CASESENSITIVE'";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,mix:title)[fn:upper-case(@jcr:title)='CASESENSITIVE']";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
```

```
QueryResult result = query.execute();
```

- LOWER case

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM mix:title WHERE LOWER(jcr:title) = 'casesensitive'";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,mix:title)[fn:lower-case(@jcr:title)='casesensitive']";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching the Result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

NodeIterator will return "document1", "document2" and "document3" (in all examples).

We can also get a table:

```
String[] columnNames = result.getColumnNames();
```

```
RowIterator rit = result.getRows();
while (rit.hasNext())
{
  Row row = rit.nextRow();
  // get values of the row
  Value[] values = row.getValues();
}
```

Table content is:

Table 28.11. Table content

jcr:title	...	jcr:path
CaseSensitive	...	/document1
casesensitive	...	/document2
caseSENSITIVE	...	/document3

Date Property Comparison

Find all nodes of primary type "nt:resource" whose jcr:lastModified property value is greater than 2006-06-04 and less than 2008-06-04.

Repository Structure

Repository contains nt:resource nodes with different values of jcr:lastModified property

- root
  - document1 (nt:file)
    - jcr:content (nt:resource) jcr:lastModified="2006-01-19T15:34:15.917+02:00"
  - document2 (nt:file)
    - jcr:content (nt:resource) jcr:lastModified="2005-01-19T15:34:15.917+02:00"
  - document3 (nt:file)
    - jcr:content (nt:resource) jcr:lastModified="2007-01-19T15:34:15.917+02:00"

Query Execution

SQL

In SQL you have to use the keyword **TIMESTAMP** for date comparisons. Otherwise, the date would be interpreted as a string. The date has to be surrounded by single quotes

(TIMESTAMP 'datetime') and in the ISO standard format: YYYY-MM-DDThh:mm:ss.sTZD (*http://en.wikipedia.org/wiki/ISO_8601* and well explained in a W3C note *http://www.w3.org/TR/NOTE-datetime*).

You will see that it can be a date only (YYYY-MM-DD) but also a complete date and time with a timezone designator (TZD).

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
StringBuffer sb = new StringBuffer();
sb.append("select * from nt:resource where ");
sb.append("( jcr:lastModified >= TIMESTAMP '");
sb.append("2006-06-04T15:34:15.917+02:00");
sb.append("' )");
sb.append(" and ");
sb.append("( jcr:lastModified <= TIMESTAMP '");
sb.append("2008-06-04T15:34:15.917+02:00");
sb.append("' )");
String sqlStatement = sb.toString();
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

Compared to the SQL format, you have to use the keyword **xs:dateTime** and surround the datetime by extra brackets: xs:dateTime('datetime'). The actual format of the datetime also conforms with the ISO date standard.

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
StringBuffer sb = new StringBuffer();
sb.append("//element(*,nt:resource)");
sb.append("[");
sb.append("@jcr:lastModified >= xs:dateTime('2006-08-19T10:11:38.281+02:00')");
sb.append(" and ");
sb.append("@jcr:lastModified <= xs:dateTime('2008-06-04T15:34:15.917+02:00')");
sb.append("]");
String xpathStatement = sb.toString();
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
```

```
QueryResult result = query.execute();
```

Fetching the result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
  Node foundNode = it.nextNode();
}
```

NodeIterator will return "/document3/jcr:content".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
  Row row = rit.nextRow();
  // get values of the row
  Value[] values = row.getValues();
}
```

The table content is:

Table 28.12. Table content

jcr:lastModified	...	jcr:path
2007-01-19T15:34:15.917+02:00	...	/document3/jcr:content

Node Name Constraint

Find all nodes with primary type 'nt:file' whose node name is 'document'. The node name is accessible by a function called "fn:name()".

> **Note**
>
> fn:name() can be used ONLY with an equal('=') comparison.

Repository Structure

The repository contains nt:file nodes with different names.

- root
  - document1 (nt:file)
  - file (nt:file)
  - somename (nt:file)

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM nt:file WHERE fn:name() = 'document'";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,nt:file)[fn:name() = 'document']";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching the Result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
```

```
}
```

The NodeIterator will return the node whose fn:name equals "document".

Also we can get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
}
```

Table content is:

Table 28.13. Table content

jcr:path	jcr:score
/document1	3575

Multivalue Property Comparison

Find all nodes with the primary type 'nt:unstructured' whose property 'multiprop' contains both values "one" and "two".

Repository Structure

The repository contains nt:unstructured nodes with different 'multiprop' properties.

- root
  - node1 (nt:unstructured) multiprop = ["one","two"]
  - node1 (nt:unstructured) multiprop = ["one","two","three"]
  - node1 (nt:unstructured) multiprop = ["one","five"]

Query Execution

SQL

```
// make SQL query
```

```
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM nt:unstructured WHERE multiprop = 'one' AND multiprop
 = 'two'";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,nt:unstructured)[@multiprop = 'one' and @multiprop =
 'two']";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching the Result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

The NodeIterator will return "node1" and "node2".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
```

```
}
```

Table content is:

Table 28.14. Table content

jcr:primarytyp	jcr:path	jcr:score
nt:unstructured	/node1	3806
nt:unstructured	/node2	3806

Exact Path Constraint

Find a node with the primary type 'nt:file' that is located on the exact path "/folder1/folder2/document1".

Repository Structure

Repository filled by different nodes. There are several folders which contain other folders and files.

- root
  - folder1 (nt:folder)
    - folder2 (nt:folder)
      - document1 (nt:file) // This document we want to find
      - folder3 (nt:folder)
        - document1 (nt:file)

Query Execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// we want find 'document1'
String sqlStatement = "SELECT * FROM nt:file WHERE jcr:path = '/folder1/folder2/document1'";
// create query
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// we want to find 'document1'
String xpathStatement = "/jcr:root/folder1[1]/folder2[1]/element(document1,nt:file)[1]";
// create query
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Remark: The indexes [1] are used in order to get the same result as the SQL statement. SQL by default only returns the first node, whereas XPath fetches by default all nodes.

Fetching the Result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

NodeIterator will return expected "document1".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
}
```

Table content is:

Table 28.15. Table content

jcr:path	jcr:score
/folder1/folder2/document1	1030

Child Node Constraint

Find all nodes with the primary type 'nt:folder' that are children of node by path "/root1/root2". Only find children, do not find further descendants.

Repository Structure

The repository is filled by "nt:folder" nodes. The nodes are placed in a multilayer tree.

- root
  - folder1 (nt:folder)
    - folder2 (nt:folder)
      - folder3 (nt:folder) // This node we want to find
        - folder4 (nt:folder) // This node is not child but a descendant of '/folder1/folder2/'.
      - folder5 (nt:folder) // This node we want to find

Query Execution

SQL

The use of "%" in the LIKE statement includes any string, therefore there is a second LIKE statement that excludes that the string contains "/". This way child nodes are included but descendant nodes are excluded.

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM nt:folder WHERE jcr:path LIKE '/folder1/folder2/%' AND
 NOT jcr:path LIKE '/folder1/folder2/%/%'";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "/jcr:root/folder1[1]/folder2[1]/element(*,nt:folder)";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
```

```
QueryResult result = query.execute();
```

Fetching the Result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

The NodeIterator will return "folder3" and "folder5".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
}
```

The table content is:

Table 28.16. Table content

jcr:path	jcr:score
/folder1/folder2/folder3	1707
/folder1/folder2/folder5	1707

Finding All Descendant Nodes

Find all nodes with the primary type 'nt:folder' that are descendants of the node "/folder1/folder2".

Repository Structure

The repository contains "nt:folder" nodes. The nodes are placed in a multilayer tree.

- root

- folder1 (nt:folder)
  - folder2 (nt:folder)
    - folder3 (nt:folder) // This node we want to find
      - folder4 (nt:folder) // This node we want to find
    - folder5 (nt:folder) // This node we want to find

Query Execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM nt:folder WHERE jcr:path LIKE '/folder1/folder2/%'";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "/jcr:root/folder1[1]/folder2[1]//element(*,nt:folder)";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching the Result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

The NodeIterator will return "folder3", "folder4" and "folder5" nodes.

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
}
```

Table content is:

Table 28.17. Table content

jcr:path	jcr:score
/folder1/folder2/folder3	1000
/folder1/folder2/folder3/folder4	1000
/folder1/folder2/folder5	1000

Sorting Nodes by Property

Select all nodes with the mixin type ''mix:title' and order them by the 'prop_pagecount' property.

Repository Structure

The repository contains several mix:title nodes, where prop_pagecount has different values.

- root
  - document1 (mix:title) jcr:title="War and peace" jcr:description="roman" prop_pagecount=4
  - document2 (mix:title) jcr:title="Cinderella" jcr:description="fairytale" prop_pagecount=7
  - document3 (mix:title) jcr:title="Puss in Boots" jcr:description="fairytale" prop_pagecount=1

Query Execution

SQL

```
// make SQL query
```

```
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM mix:title ORDER BY prop_pagecount ASC";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,mix:title) order by @prop_pagecount ascending";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching the Result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

The NodeIterator will return nodes in the following order "document3", "document1", "document2".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
}
```

Table content is:

Table 28.18. Table content

jcr:title	jcr:description	prop_pagecount	jcr:path	jcr:score
Puss in Boots	fairytale	1	/document3	1405
War and peace	roman	4	/document1	1405
Cinderella	fairytale	7	/document2	1405

Ordering by Descendant Nodes Property (XPath only)

Find all nodes with the primary type 'nt:unstructured' and sort them by the property value of descendant nodes with the relative path '/a/b'.

Note

This ORDER BY construction only works in XPath!

Repository structure:

- root
  - node1 (nt:unstructured)
    - a (nt:unstructured)
      - b (nt:unstructured)
  - node2 (nt:unstructured)
    - a (nt:unstructured)
      - b (nt:unstructured)
        - c (nt:unstructured) prop = "a"
  - node3 (nt:unstructured)
    - a (nt:unstructured)
      - b (nt:unstructured)
        - c (nt:unstructured) prop = "b"

Query Execution

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "/jcr:root/* order by a/b/c/@prop descending;
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching the Result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

NodeIterator will return nodes in the following order - "node3","node2" and "node1".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
}
```

Table content is:

Table 28.19. Table content

jcr:primaryType	jcr:path	jcr:score
nt:unstructured	/testroot/node3	1000
nt:unstructured	/testroot/node2	1000
nt:unstructured	/testroot/node1	1000

Ordering by Score

Select all nodes with the mixin type 'mix:title' containing any word from the set {'brown','fox','jumps'}. Then, sort result by the score in ascending node. This way nodes that match better the query statement are ordered at the last positions in the result list.

Info

SQL and XPath queries support both score constructions jcr:score and jcr:score()

```
SELECT * FROM nt:base ORDER BY jcr:score [ASC|DESC]
SELECT * FROM nt:base ORDER BY jcr:score()[ASC|DESC]

//element(*,nt:base) order by jcr:score() [descending]
//element(*,nt:base) order by @jcr:score [descending]
```

Do not use "ascending" combined with jcr:score in XPath. The following XPath statement may throw an exception:

```
... order by jcr:score() ascending
```

Do not set any ordering specifier - ascending is default:

```
... order by jcr:score()
```

Repository Structure

The repository contains mix:title nodes, where the jcr:description has different values.

- root
  - document1 (mix:title) jcr:description="The quick brown fox jumps over the lazy dog."
  - document2 (mix:title) jcr:description="The brown fox lives in the forest."
  - document3 (mix:title) jcr:description="The fox is a nice animal."

Query Execution

SQL

```
// make SQL query
```

```
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM mix:title WHERE CONTAINS(*, 'brown OR fox OR
 jumps') ORDER BY jcr:score() ASC";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,mix:title)[jcr:contains(., 'brown OR fox OR jumps')] order by
 jcr:score()";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching the Result

Let's get nodes

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

NodeIterator will return nodes in the following order: "document3", "document2", "document1".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
```

```
}
```

Table content is:

Table 28.20. Table content

jcr:description	...	jcr:path	jcr:score
The fox is a nice animal.	...	/document3	2512
The brown fox lives in the forest.	...	/document2	3595
The quick brown fox jumps over the lazy dog.	...	/document1	5017

Ordering by Path or Name

Ordering by jcr:path or jcr:name does not supported.

There is two ways to order results, when path may be used as criteria:

- Order by property with value type NAME or PATH (jcr supports it)
- Order by jcr:path or jcr:name - sort by exact path or name of node (jcr do not supports it)

If no order specification is supplied in the query statement, implementations may support document order on the result nodes (see jsr-170 / 6.6.4.2 Document Order). And it's sorted by order number.

By default, (if query do not contains any ordering statements) result nodes is sorted by document order.

```
SELECT * FROM nt:unstructured WHERE jcr:path LIKE 'testRoot/%'
```

Fulltext Search by Property

Find all nodes containing a mixin type 'mix:title' and whose 'jcr:description' contains "forest" string.

Repository Structure

The repository is filled with nodes of the mixin type 'mix:title' and different values of the 'jcr:description' property.

- root

- document1 (mix:title) jcr:description = "The quick brown fox jumps over the lazy dog."
- document2 (mix:title) jcr:description = "The brown fox lives in a *forest*." // This is the node we want to find
- document3 (mix:title) jcr:description = "The fox is a nice animal."
- document4 (nt:unstructured) jcr:description = "There is the word forest, too."

Query Execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// we want find document which contains "forest" word
String sqlStatement = "SELECT \* FROM mix:title WHERE CONTAINS(jcr:description, 'forest')";
// create query
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// we want find document which contains "forest" word
String xpathStatement = "//element(*,mix:title)[jcr:contains(@jcr:description, 'forest')]";
// create query
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching the Result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
```

```
}
```

NodeIterator will return "document2".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
  Row row = rit.nextRow();
  // get values of the row
  Value[] values = row.getValues();
}
```

Table content is:

Table 28.21. Table content

jcr:description	...	jcr:path
The brown fox lives in forest.	...	/document2

Fulltext Search by All Properties in Node

Find nodes with mixin type 'mix:title' where any property contains 'break' string.

Repository structure:

Repository filled with different nodes with mixin type 'mix:title' and different values of 'jcr:title' and 'jcr:description' properties.

- root
  - document1 (mix:title) jcr:title ='Star Wars' jcr:description = 'Dart rules!!'
  - document2 (mix:title) jcr:title ='Prison *break*' jcr:description = 'Run, Forest, run))'
  - document3 (mix:title) jcr:title ='Titanic' jcr:description = 'An iceberg *break*s a ship.'

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
```

```
String sqlStatement = "SELECT * FROM mix:title WHERE CONTAINS(*,'break')";
// create query
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// we want find 'document1'
String xpathStatement = "//element(*,mix:title)[jcr:contains(.,'break')]";
// create query
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Let's get nodes:

```
NodeIterator it = result.getNodes();

while(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

NodeIterator will return "document1" and "document2".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
}
```

Table content is:

Table 28.22. Table content

jcr:title	jcr:description	...	jcr:path
Prison break.	Run, Forest, run))	...	/document2
Titanic	An iceberg breaks a ship.	...	/document3

Ignoring Accent Symbols. New Analyzer Setting.

In this example, we will create new Analyzer, set it in QueryHandler configuration, and make query to check it.

Standard analyzer does not normalize accents like é,è,à. So, a word like 'tréma' will be stored to index as 'tréma'. But if we want to normalize such symbols or not? We want to store 'tréma' word as 'trema'.

There is two ways of setting up new Analyzer (no matter standarts or our):

- The first way: Create descendant class of SearchIndex with new Analyzer (see *Search Configuration*);

There is only one way - create new Analyzer (if there is no previously created and accepted for our needs) and set it in Search index.

- The second way: Register new Analyzer in QueryHandler configuration (this one eccepted since 1.12 version);

We will use the last one:

- Create new MyAnalyzer

```
public class MyAnalyzer extends Analyzer
{
  @Override
  public TokenStream tokenStream(String fieldName, Reader reader)
  {
    StandardTokenizer tokenStream = new StandardTokenizer(reader);
    // process all text with standard filter
    // removes 's (as 's in "Peter's") from the end of words and removes dots from acronyms.
    TokenStream result = new StandardFilter(tokenStream);
```

```
    // this filter normalizes token text to lower case
    result = new LowerCaseFilter(result);
     // this one replaces accented characters in the ISO Latin 1 character set (ISO-8859-1) by
 their unaccented equivalents
    result = new ISOLatin1AccentFilter(result);
    // and finally return token stream
    return result;
  }
}
```

- Then, register new MyAnalyzer in configuration

```
<workspace name="ws">
  ...
  <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
    <properties>
      <property name="analyzer" value="org.exoplatform.services.jcr.impl.core.MyAnalyzer"/>
      ...
    </properties>
  </query-handler>
  ...
</workspace>
```

After that, check it with query:

Find node with mixin type 'mix:title' where 'jcr:title' contains "tréma" and "naïve" strings.

Repository structure:

Repository filled by nodes with mixin type 'mix:title' and different values of 'jcr:title' property.

- root
  - node1 (mix:title) jcr:title = "tréma blabla naïve"
  - node2 (mix:title) jcr:description = "trema come text naive"

Query execution

SQL

```
// make SQL query
```

```
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM mix:title WHERE CONTAINS(jcr:title, 'tr\u00E8ma na\u00EFve')";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,mix:title)[jcr:contains(@jcr:title, 'tr\u00E8ma na\u00EFve')]";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

NodeIterator will return "node1" and "node2". How is it possible? Remember that our MyAnalyzer transforms 'tréma' word to 'trema'. So node2 accepts our constraints to.

Also, we can get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
```

```
}
```

Table content is

Table 28.23. Table content

cr:title	...	cr:path
trèma blabla naïve	...	/node1
trema come text naive	...	/node2

Finding nt:file node by content of child jcr:content node

The node type nt:file represents a file. It requires a single child node, called jcr:content. This node type represents images and other binary content in a JCRWiki entry. The node type of jcr:conent is nt:resource which represents the actual content of a file.

Find node with the primary type is 'nt:file' and which whose 'jcr:content' child node contains "cats".

Normally, we can't find nodes (in our case) using just JCR SQL or XPath queries. But we can configure indexing so that nt:file aggregates jcr:content child node.

So, change indexing-configuration.xml:

```
<?xml version="1.0"?>
<!DOCTYPE configuration SYSTEM "http://www.exoplatform.org/dtd/indexing-configuration-1.2.dtd">
<configuration xmlns:jcr="http://www.jcp.org/jcr/1.0"
               xmlns:nt="http://www.jcp.org/jcr/nt/1.0">
   <aggregate primaryType="nt:file">
      <include>jcr:content</include>
      <include>jcr:content/*</include>
      <include-property>jcr:content/jcr:lastModified</include-property>
   </aggregate>
</configuration>
```

Now the content of 'nt:file' and 'jcr:content' ('nt:resource') nodes are concatenated in a single Lucene document. Then, we can make a fulltext search query by content of 'nt:file'; this search includes the content of child 'jcr:content' node.

Repository structure:

Repository contains different nt:file nodes.

- root

- document1 (nt:file)
  - jcr:content (nt:resource) jcr:data = "The quick brown fox jumps over the lazy dog."
- document2 (nt:file)
  - jcr:content (nt:resource) jcr:data = "Dogs do not like cats."
- document3 (nt:file)
  - jcr:content (nt:resource) jcr:data = "Cats jumping high."

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM nt:file WHERE CONTAINS(*,'cats')";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,nt:file)[jcr:contains(.,'cats')]";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching the result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
```

```
}
```

NodeIterator will return "document2" and "document3".

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
   Row row = rit.nextRow();
   // get values of the row
   Value[] values = row.getValues();
}
```

Table content is:

Table 28.24. Table content

jcr:path	jcr:score
/document2	1030
/document3	1030

Changing Priority of Node

In this example, we will set different boost values for predefined nodes, and will check effect by selecting those nodes and order them by jcr:score.

The default boost value is 1.0. Higher boost values (a reasonable range is 1.0 - 5.0) will yield a higher score value and appear as more relevant.

Note

See 4.2.2 Index Boost Value *Search Configuration*

Indexing configuration

In next configuration, we will set boost values for nt:ustructured nodes 'text' property.

indexing-config.xml:

```
<!--
This rule actualy do nothing. 'text' property has default boost value.
-->
```

```
<index-rule nodeType="nt:unstructured" condition="@rule='boost1'">
  <!-- default boost: 1.0 -->
  <property>text</property>
</index-rule>

<!--
Set boost value as 2.0 for 'text' property in nt:unstructured nodes where property 'rule' equal to
 'boost2'
-->
<index-rule nodeType="nt:unstructured" condition="@rule='boost2'">
  <!-- boost: 2.0 -->
  <property boost="2.0">text</property>
</index-rule>

<!--
Set boost value as 3.0 for 'text' property in nt:unstructured nodes where property 'rule' equal to
 'boost3'
-->
<index-rule nodeType="nt:unstructured" condition="@rule='boost3'">
  <!-- boost: 3.0 -->
  <property boost="3.0">text</property>
</index-rule>
```

Repository structure:

Repository contains many nodes with primary type nt:unstructured. Each node contains 'text' property and 'rule' property with different values.

- root
  - node1(nt:unstructured) rule='boost1' text='The quick brown fox jump...'
  - node2(nt:unstructured) rule='boost2' text='The quick brown fox jump...'
  - node3(nt:unstructured) rule='boost3' text='The quick brown fox jump...'

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM nt:unstructured WHERE CONTAINS(text, 'quick')
 ORDER BY jcr:score() DESC";
```

```
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,nt:unstructured)[jcr:contains(@text, 'quick')] order by @jcr:score descending";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
   Node findedNode = it.nextNode();
}
```

NodeIterator will return nodes in next order "node3", "node2", "node1".

Removing Nodes Property From Indexing Scope

In this example, we will exclude some 'text' property of nt:unstructured node from indexind. And, therefore, node will not be found by the content of this property, even if it accepts all constraints.

First of all, add rules to indexing-configuration.xml:

```
<index-rule nodeType="nt:unstructured" condition="@rule='nsiTrue'">
    <!-- default value for nodeScopeIndex is true -->
    <property>text</property>
</index-rule>

<index-rule nodeType="nt:unstructured" condition="@rule='nsiFalse'">
```

```
    <!-- do not include text in node scope index -->
    <property nodeScopeIndex="false">text</property>
</index-rule>
```

Note

See *Search Configuration*

Repository structure:

Repository contains nt:unstructured nodes, with same 'text'property and different 'rule' properties (even null)

- root
  - node1 (nt:unstructured) rule="nsiTrue" text="The quick brown fox ..."
  - node2 (nt:unstructured) rule="nsiFalse" text="The quick brown fox ..."
  - node3 (nt:unstructured) text="The quick brown fox ..." // as you see this node not mentioned in indexing-coniguration

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM nt:unstructured WHERE CONTAINS(*,'quick')";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,nt:unstructured)[jcr:contains(., 'quick')]";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
  Node findedNode = it.nextNode();
}
```

NodeIterator will return "node1" and "node3". Node2, as you see, is not in result set.

Also, we can get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
  Row row = rit.nextRow();
  // get values of the row
  Value[] values = row.getValues();
}
```

Table content is

Table 28.25. Table content

jcr:primarytype	jcr:path	jcr:score
nt:unstructured	/node1	3806
nt:unstructured	/node3	3806

Regular Expression as Property Name in Indexing Rules

In this example, we want to configure indexind in the next way. All properties of nt:unstructured nodes must be excluded from search, except properties whoes names ends with 'Text' string. First of all, add rules to indexing-configuration.xml:

```
<index-rule nodeType="nt:unstructured"">
  <property isRegexp="true">.*Text</property>
</index-rule>
```

Note

See *Search Configuration*

Now, let's check this rule with simple query - select all nodes with primary type 'nt:unstructured' and containing 'quick' string (fulltext search by full node).

Repository structure:

Repository contains nt:unstructured nodes, with different 'text'-like named properties

- root
  - node1 (nt:unstructured) Text="The quick brown fox ..."
  - node2 (nt:unstructured) OtherText="The quick brown fox ..."
  - node3 (nt:unstructured) Textle="The quick brown fox ..."

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM nt:unstructured WHERE CONTAINS(*,'quick')";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,nt:unstructured)[jcr:contains(., 'quick')]";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
  Node findedNode = it.nextNode();
}
```

NodeIterator will return "node1" and "node2". "node3", as you see, is not in result set.

Also, we can get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
  Row row = rit.nextRow();
  // get values of the row
  Value[] values = row.getValues();
}
```

Table content is:

Table 28.26. Table content

jcr:primarytype	jcr:path	jcr:score
nt:unstructured	/node1	3806
nt:unstructured	/node2	3806

High-lighting Result of Fulltext Search

It's also called excerption (see Excerpt configuration in *Search Configuration* and in *Searching Repository* article).

The goal of this query is to find words "eXo" and "implementation" with fulltext search and high-light this words in result value.

Base info

High-lighting is not default feature so we must set it in jcr-config.xml, also excerpt provider must be defined:

```
<query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
```

```
  <properties>
    ...
    <property name="support-highlighting" value="true" />
                                                    <property          name="excerptprovider-class"
 value="org.exoplatform.services.jcr.impl.core.query.lucene.WeightedHTMLExcerpt"/>
    ...
  <properties>
</query-handler>
```

Also, remember that we can make indexing rules, as in the example below:

Let's write rule for all nodes with primary node type 'nt:unstructed' where property 'rule' equal to "excerpt" string. For those nodes, we will exclude property "title" from high-lighting and set "text" property as highlightable. Indexing-configuration.xml must containt the next rule:

```
<index-rule nodeType="nt:unstructured" condition="@rule='excerpt'">
  <property useInExcerpt="false">title</property>
  <property>text</property>
</index-rule>
```

Repository structure:

We have single node with primary type 'nt:unstructured'

- document (nt:unstructured)
  - rule = "excerpt"
  - title = "eXoJCR"
  - text = "eXo is a JCR implementation"

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT rep:excerpt() FROM nt:unstructured WHERE CONTAINS(*, 'eXo
 implementation')";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
```

```
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,nt:unstructured)[jcr:contains(., 'eXo implementation')]/rep:excerpt(.)";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Now let's see on the result table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
  Row row = rit.nextRow();
  // get values of the row
  Value[] values = row.getValues();
}
```

Table content is

Table 28.27. Table content

rep:excerpt()	jcr:path	jcr:score
\<div\>\<span\>\<strong\>eXo\</strong\> is a JCR \<strong\>implementation\</strong\>\</span\>\</div\>	//testroot/node1	335

As you see, words "eXo" and "implamentation" is highlighted.

Also, we can get exactly "rep:excerpt" value:

```
RowIterator rows = result.getRows();
Value excerpt = rows.nextRow().getValue("rep:excerpt(.)");
```

```
// excerpt will be equal to "<div><span\><strong>eXo</strong> is a JCR
 <strong>implementation</strong></span></div>"
```

Searching By Synonim

Find all mix:title nodes where title contains synonims to 'fast' word.

Note

See also about synonim propvider configuration - *Searching Repository Content*

Synonim provider must be configured in indexing-configuration.xml :

```
<query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
  <properties>
    ...
                                                <property name="synonymprovider-class"
 value="org.exoplatform.services.jcr.impl.core.query.lucene.PropertiesSynonymProvider" />
    <property name="synonymprovider-config-path" value="../../synonyms.properties" />
    ...
  </properties>
</query-handler>
```

File synonim.properties contains next synonims list:

```
ASF=Apache Software Foundation
quick=fast
sluggish=lazy
```

Repository structure:

Repository contains mix:title nodes, where jcr:title has different values.

- root
  - document1 (mix:title) jcr:title="The quick brown fox jumps over the lazy dog."

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
```

```
// create query
String sqlStatement = "SELECT * FROM mix:title WHERE CONTAINS(jcr:title, '~fast')";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*,mix:title)[jcr:contains(@jcr:title, '~fast')]";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Let's get nodes:

```
NodeIterator it = result.getNodes();

if(it.hasNext())
{
  Node findedNode = it.nextNode();
}
```

NodeIterator will return expected document1. This is a purpose of synonim providers. Find by specified word, but return by all synonims to.

Checking the spelling of Phrase

Check the correct spelling of phrase 'quik OR (-foo bar)' according to data already stored in index.

Note

See also about SpellChecker configuration - *Searching Repository Content*

SpellChecker must be settled in query-handler config.

test-jcr-config.xml:

```
<query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
```

```
  <properties>
    ...
                                                <property                name="spellchecker-class"

/>
    ...
  </properties>
</query-handler>
```

Repository structure:

Repository contains node, with string property "The quick brown fox jumps over the lazy dog."

- root
  - node1 property="The quick brown fox jumps over the lazy dog."

Query execution

Query looks only for root node, because spell checker looks for suggestions by full index. So complicated query is redundant.

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT rep:spellcheck() FROM nt:base WHERE jcr:path = '/' AND
 SPELLCHECK('quik OR (-foo bar)')";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "/jcr:root[rep:spellcheck('quik OR (-foo bar)')]/(rep:spellcheck())";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Get suggestion of coorect spelling our phrase:

```
RowIterator it = result.getRows();
Row r = rows.nextRow();
Value v = r.getValue("rep:spellcheck()");
String correctPhrase = v.getString();
```

So, correct spelling for phrase "quik OR (-foo bar)" is "quick OR (-fox bar)".

Finding Similar Nodes

Find similar nodes to node by path '/baseFile/jcr:content'.

In our example, baseFile will contain text where "terms" word happens many times. That's a reason why the existanse of this word will be used as a criteria of node similarity (for node baseFile).

Note

See also about Similarity and configuration - *Searching Repository Content*

Higlighting support must be added to configuration. test-jcr-config.xml:

```
<query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
  <properties>
    ...
    <property name="support-highlighting" value="true" />
    ...
  </properties>
</query-handler>
```

Repository structure:

Repository contains many nt:file nodes"

- root
  - baseFile (nt:file)
    - jcr:content (nt:resource) jcr:data="Similarity is determined by looking up **terms** that are common to nodes. There are some conditions that must be met for a **term** to be considered. This is required to limit the number possibly relevant **terms**. Only **terms** with at least 4

characters are considered. Only **terms** that occur at least 2 times in the source node are considered. Only **terms** that occur in at least 5 nodes are considered."

- target1 (nt:file)
  - jcr:content (nt:resource) jcr:data="Similarity is determined by looking up **terms** that are common to nodes."
- target2 (nt:file)
  - jcr:content (nt:resource) jcr:data="There is no you know what"
- target3 (nt:file)
  - jcr:content (nt:resource) jcr:data=" **Terms** occures here"

Query execution

SQL

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
// create query
String sqlStatement = "SELECT * FROM nt:resource WHERE SIMILAR(.,'/baseFile/jcr:content')";
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

XPath

```
// make XPath query
QueryManager queryManager = workspace.getQueryManager();
// create query
String xpathStatement = "//element(*, nt:resource)[rep:similar(., '/testroot/baseFile/jcr:content')]";
Query query = queryManager.createQuery(xpathStatement, Query.XPATH);
// execute query and fetch result
QueryResult result = query.execute();
```

Fetching result

Let's get nodes:

```
NodeIterator it = result.getNodes();
```

```
if(it.hasNext())
{
  Node findedNode = it.nextNode();
}
```

NodeIterator will return "/baseFile/jcr:content","/target1/jcr:content" and "/target3/jcr:content".

As you see the base node are also in result set.

We can also get a table:

```
String[] columnNames = result.getColumnNames();
RowIterator rit = result.getRows();
while (rit.hasNext())
{
  Row row = rit.nextRow();
  // get values of the row
  Value[] values = row.getValues();
}
```

The table content is

Table 28.28. Table content

jcr:path	...	jcr:score
/baseFile/jcr:content	...	2674
/target1/jcr:content	...	2674
/target3/jcr:content	...	2674

Tips and tricks

XPath queries containing node names starting with a number

If you execute an XPath request like this:

XPath

```
// get QueryManager
QueryManager queryManager = workspace.getQueryManager();
// make XPath query
Query query = queryManager.createQuery("/jcr:root/Documents/Publie/2010//element(*,
 exo:article)", Query.XPATH);
```

You will have an error : "Invalid request". This happens because XML does not allow names starting with a number - and XPath is part of XML: *http://www.w3.org/TR/REC-xml/#NT-Name*

Therefore, you cannot do XPath requests using a node name that starts with a number.

Easy workarounds:

- Use an SQL request.
- Use escaping :

XPath

```
// get QueryManager
QueryManager queryManager = workspace.getQueryManager();
// make XPath query
Query query = queryManager.createQuery("/jcr:root/Documents/Publie/_x0032_010//element(*,
 exo:article)", Query.XPATH);
```

Searching Repository Content

Introduction

You can find the JCR configuration file here: .../portal/WEB-INF/conf/jcr/repository-configuration.xml. Please read also *Search Configuration* for more information about index configuration.

Bi-directional RangeIterator (since 1.9)

QueryResult.getNodes() will return bi-directional NodeIterator implementation.

Note

Bi-directional NodeIterator is **not supported** in two cases:

- SQL query: select * from nt:base
- XPath query: //* .

TwoWayRangeIterator interface:

```
/**
 * Skip a number of elements in the iterator.
 *
 * @param skipNum the non-negative number of elements to skip
 * @throws java.util.NoSuchElementException if skipped past the first element
 *           in the iterator.
 */
public void skipBack(long skipNum);
```

Usage:

```
NodeIterator iter = queryResult.getNodes();
while (iter.hasNext()) {
  if (skipForward) {
    iter.skip(10); // Skip 10 nodes in forward direction
  } else if (skipBack) {
    TwoWayRangeIterator backIter = (TwoWayRangeIterator) iter;
    backIter.skipBack(10); // Skip 10 nodes back
  }
  .....
}
```

Fuzzy Searches (since 1.0)

JCR supports such features as Lucene Fuzzy Searches *Apache Lucene - Query Parser Syntax* [http://lucene.apache.org/java/2_3_2/queryparsersyntax.html].

To use it, you have to form a query like the one described below:

```
QueryManager qman = session.getWorkspace().getQueryManager();
Query q = qman.createQuery("select * from nt:base where contains(field, 'ccccc~')", Query.SQL);
QueryResult res = q.execute();
```

SynonymSearch (since 1.9)

Searching with synonyms is integrated in the jcr:contains() function and uses the same syntax as synonym searches in Google. If a search term is prefixed by a tilde symbol (~), also synonyms of the search term are taken into consideration. For example:

```
SQL: select * from nt:resource where contains(., '~parameter')

XPath: //element(*, nt:resource)[jcr:contains(., '~parameter')
```

This feature is disabled by default and you need to add a configuration parameter to the query-handler element in your jcr configuration file to enable it.

```
<param  name="synonymprovider-config-path" value="..you path to configuration file....."/>
<param                                                    name="synonymprovider-class"
 value="org.exoplatform.services.jcr.impl.core.query.lucene.PropertiesSynonymProvider"/>
```

```
/**
 * <code>SynonymProvider</code> defines an interface for a component that
 * returns synonyms for a given term.
 */
public interface SynonymProvider {

   /**
    * Initializes the synonym provider and passes the file system resource to
    * the synonym provider configuration defined by the configuration value of
    * the <code>synonymProviderConfigPath</code> parameter. The resource may be
    * <code>null</code> if the configuration parameter is not set.
    *
```

```
    * @param fsr the file system resource to the synonym provider
    *            configuration.
    * @throws IOException if an error occurs while initializing the synonym
    *                     provider.
    */
   public void initialize(InputStream fsr) throws IOException;

   /**
    * Returns an array of terms that are considered synonyms for the given
    * <code>term</code>.
    *
    * @param term a search term.
    * @return an array of synonyms for the given <code>term</code> or an empty
    *         array if no synonyms are known.
    */
   public String[] getSynonyms(String term);
}
```

High-lighting (Since 1.9)

An ExcerptProvider retrieves text excerpts for a node in the query result and marks up the words in the text that match the query terms.

By default highlighting words matched the query is disabled because this feature requires that additional information is written to the search index. To enable this feature, you need to add a configuration parameter to the query-handler element in your jcr configuration file to enable it.

```
<param name="support-highlighting" value="true"/>
```

Additionally, there is a parameter that controls the format of the excerpt created. In JCR 1.9, the default is set to org.exoplatform.services.jcr.impl.core.query.lucene.DefaultHTMLExcerpt. The configuration parameter for this setting is:

```
<param                                                    name="excerptprovider-class"
 value="org.exoplatform.services.jcr.impl.core.query.lucene.DefaultXMLExcerpt"/>
```

DefaultXMLExcerpt

This excerpt provider creates an XML fragment of the following form:

```
<excerpt>
```

```
   <fragment>
      <highlight>exoplatform</highlight> implements both the mandatory
      XPath and optional SQL <highlight>query</highlight> syntax.
   </fragment>
   <fragment>
      Before parsing the XPath <highlight>query</highlight> in
      <highlight>exoplatform</highlight>, the statement is surrounded
   </fragment>
</excerpt>
```

DefaultHTMLExcerpt

This excerpt provider creates an HTML fragment of the following form:

```
<div>
   <span>
      <strong>exoplatform</strong> implements both the mandatory XPath
      and optional SQL <strong>query</strong> syntax.
   </span>
   <span>
      Before parsing the XPath <strong>query</strong> in
      <strong>exoplatform</strong>, the statement is surrounded
   </span>
</div>
```

How to use it

If you are using XPath, you must use the rep:excerpt() function in the last location step, just like you would select properties:

```
QueryManager qm = session.getWorkspace().getQueryManager();
Query q = qm.createQuery("//*[jcr:contains(., 'exoplatform')]/(@Title|rep:excerpt(.))",
 Query.XPATH);
QueryResult result = q.execute();
for (RowIterator it = result.getRows(); it.hasNext(); ) {
   Row r = it.nextRow();
   Value title = r.getValue("Title");
   Value excerpt = r.getValue("rep:excerpt(.)");
}
```

The above code searches for nodes that contain the word exoplatform and then gets the value of the Title property and an excerpt for each result node.

It is also possible to use a relative path in the call Row.getValue() while the query statement still remains the same. Also, you may use a relative path to a string property. The returned value will then be an excerpt based on string value of the property.

Both available excerpt provider will create fragments of about 150 characters and up to 3 fragments.

In SQL, the function is called excerpt() without the rep prefix, but the column in the RowIterator will nonetheless be labled rep:excerpt(.)!

```
QueryManager qm = session.getWorkspace().getQueryManager();
Query q = qm.createQuery("select excerpt(.) from nt:resource where contains(., 'exoplatform')",
 Query.SQL);
QueryResult result = q.execute();
for (RowIterator it = result.getRows(); it.hasNext(); ) {
   Row r = it.nextRow();
   Value excerpt = r.getValue("rep:excerpt(.)");
}
```

SpellChecker

The lucene based query handler implementation supports a pluggable spell checker mechanism. By default, spell checking is not available and you have to configure it first. See parameter spellCheckerClass on page *Search Configuration.* JCR currently provides an implementation class , which uses the *lucene-spellchecker* [http://wiki.apache.org/jakarta-lucene/SpellChecker] to contribute . The dictionary is derived from the fulltext indexed content of the workspace and updated periodically. You can configure the refresh interval by picking one of the available inner classes of org.exoplatform.services.jcr.impl.core.query.lucene.spell.LuceneSpellChecker:

- OneMinuteRefreshInterval
- FiveMinutesRefreshInterval
- ThirtyMinutesRefreshInterval
- OneHourRefreshInterval
- SixHoursRefreshInterval
- TwelveHoursRefreshInterval
- OneDayRefreshInterval

For example, if you want a refresh interval of six hours, the class name is: org.exoplatform.services.jcr.impl.core.query.lucene.spell.LuceneSpellChecker$SixHoursRefreshInterval. If you use org.exoplatform.services.jcr.impl.core.query.lucene.spell.LuceneSpellChecker, the refresh interval will be one hour.

The spell checker dictionary is stored as a lucene index under **"index-dir"/spellchecker**. If it does not exist, a background thread will create it on startup. Similarly, the dictionary refresh is also done in a background thread to not block regular queries.

How do I use it?

You can spell check a fulltext statement either with an XPath or a SQL query:

```
// rep:spellcheck('explatform') will always evaluate to true
Query query = qm.createQuery("/jcr:root[rep:spellcheck('explatform')]/(rep:spellcheck())",
 Query.XPATH);
RowIterator rows = query.execute().getRows();
// the above query will always return the root node no matter what string we check
Row r = rows.nextRow();
// get the result of the spell checking
Value v = r.getValue("rep:spellcheck()");
if (v == null) {
   // no suggestion returned, the spelling is correct or the spell checker
   // does not know how to correct it.
} else {
   String suggestion = v.getString();
}
```

And the same using SQL:

```
// SPELLCHECK('exoplatform') will always evaluate to true
Query query = qm.createQuery("SELECT rep:spellcheck() FROM nt:base WHERE jcr:path = '/'
 AND SPELLCHECK('explatform')", Query.SQL);
RowIterator rows = query.execute().getRows();
// the above query will always return the root node no matter what string we check
Row r = rows.nextRow();
// get the result of the spell checking
Value v = r.getValue("rep:spellcheck()");
if (v == null) {
   // no suggestion returned, the spelling is correct or the spell checker
   // does not know how to correct it.
} else {
```

```
  String suggestion = v.getString();
}
```

Similarity (Since 1.12)

Starting with version, 1.12 JCR allows you to search for nodes that are similar to an existing node.

Similarity is determined by looking up terms that are common to nodes. There are some conditions that must be met for a term to be considered. This is required to limit the number possibly relevant terms.

- Only terms with at least 4 characters are considered.
- Only terms that occur at least 2 times in the source node are considered.
- Only terms that occur in at least 5 nodes are considered.

Note: The similarity functionality requires that the support Hightlighting is enabled. Please make sure that you have the following parameter set for the query handler in your workspace.xml.

```
<param name="support-highlighting" value="true"/>
```

The functions are called rep:similar() (in XPath) and similar() (in SQL) and have two arguments:

relativePath: a relative path to a descendant node or . for the current node. absoluteStringPath: a string literal that contains the path to the node for which to find similar nodes.

> Relative path is not supported yet.

Examples:

```
//element(*, nt:resource)[rep:similar(., '/parentnode/node.txt/jcr:content')]
```

Finds nt:resource nodes, which are similar to node by path /parentnode/node.txt/jcr:content.

Fulltext Search And Affecting Settings

Property content indexing

Each property of a node (if it is indexable) is processed with Lucene analyzer and stored in Lucene index. That's called indexing of a property. After that we can perform a fulltext search among these indexed properties.

Lucene Analyzers

The sense of analyzers is to transform all strings stored in the index in a well-defined condition. The same analyzer(s) is/are used when searching in order to adapt the query string to the index reality.

Therefore, performing the same query using different analyzers can return different results.

Now, let's see how the same string is transformed by different analyzers.

Table 30.1. "The quick brown fox jumped over the lazy dogs"

Analyzer	Parsed
org.apache.lucene.analysis.WhitespaceAnalyzer	[The] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dogs]
org.apache.lucene.analysis.SimpleAnalyzer	[the] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dogs]
org.apache.lucene.analysis.StopAnalyzer	[quick] [brown] [fox] [jumped] [over] [lazy] [dogs]
org.apache.lucene.analysis.standard.StandardAnalyzer	[quick] [brown] [fox] [jumped] [over] [lazy] [dogs]
org.apache.lucene.analysis.snowball.SnowballAnalyzer	[quick] [brown] [fox] [jump] [over] [lazi] [dog]
org.apache.lucene.analysis.standard.StandardAnalyzer (configured without stop word - jcr default analyzer)	[the] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dogs]

Table 30.2. "XY&Z Corporation - xyz@example.com"

Analyzer	Parsed
org.apache.lucene.analysis.WhitespaceAnalyzer	[XY&Z] [Corporation] [-] [xyz@example.com]
org.apache.lucene.analysis.SimpleAnalyzer	[xy] [z] [corporation] [xyz] [example] [com]
org.apache.lucene.analysis.StopAnalyzer	[xy] [z] [corporation] [xyz] [example] [com]

Analyzer	Parsed
org.apache.lucene.analysis.standard.StandardAnalyzer	[xyz@example.com] [corporation] [xyz@example] [com]
org.apache.lucene.analysis.snowball.SnowballAnalyzer	[xyz@example.com] [corpor] [xyz@exampl] [com]
org.apache.lucene.analysis.standard.StandardAnalyzer (configured without stop word - jcr default analyzer)	[xyz@example.com] [corporation] [xyz@example] [com]

Note

StandardAnalyzer is the default analyzer in exo's jcr search engine. But we do not use stop words.

You can assign your analyzer as described in *Search Configuration*

How are different properties indexed?

Different properties are indexed in different ways, this affects to if it can be searched like fulltext by property or not.

Only two property types are indexed as fulltext searcheable: STRING and BINARY.

Table 30.3. Fulltext search by different properties

Property Type	Fulltext search by all properties	Fulltext search by exact property
STRING	YES	YES
BINARY	YES	NO

For example, #e have property jcr:data (it' BINARY). It's stored well, but you will never find any string with query like:

```
SELECT * FROM nt:resource WHERE CONTAINS(jcr:data, 'some string')
```

Because, BINARY is not searchable by fulltext search on exact property.

But, next query will return result (off course if node has searched data):

```
SELECT * FROM nt:resource WHERE CONTAINS( * , 'some string')
```

Fulltext search query examples

- *JCR.Fulltext Search by Property*

- *JCR.Fulltext Search by All Properties*
- *Find nt:file document by content of its child jcr:content node*
- *How to set a new analyzer. Accent symbols ignoring*

Different analyzers in action

First of all, we will fill repository by nodes with mixin type 'mix:title' and different values of 'jcr:description' property.

- root
  - document1 (mix:title) jcr:description = "The quick brown fox jumped over the lazy dogs"
  - document2 (mix:title) jcr:description = "Brown fox live in forest."
  - document3 (mix:title) jcr:description = "Fox is a nice animal."

Let's see analyzers effect closer. In first case, we use base jcr settings, so, as mentioned above, string "The quick brown fox jumped over the lazy dogs" will be transformed to set {[the] [quick] [brown] [fox] [jumped] [over] [the] [lazy] [dogs] }

```
// make SQL query
QueryManager queryManager = workspace.getQueryManager();
String sqlStatement = "SELECT * FROM mix:title WHERE CONTAINS(jcr:description, 'the')";
// create query
Query query = queryManager.createQuery(sqlStatement, Query.SQL);
// execute query and fetch result
QueryResult result = query.execute();
```

NodeIterator will return "document1".

Now change the default analyzer to org.apache.lucene.analysis.StopAnalyzer. Fill repository again (new Analyzer must process nodes properties) and run the same query again. It will return nothing, because stop words like "the" will be excluded from parsed string set.

JCR API Extensions

"Lazy" child nodes iteration

Concept

eXo JCR implementation offers new extended feature beyond JCR specification. Sometimes it happens that one JCR Node has hundreds or even thousands of child nodes. This situation is highly not recommended for content repository data storage, but some times it occurs. JCR Team is pleased to announce new feature that will help to have a deal with huge child lists. They can be iterated in a "lazy" manner now giving improvement in term of performance and RAM usage.

API and usage

Lazy child nodes iteration feature is accessible via extended interface org.exoplatform.services.jcr.core.ExtendedNode, the inheritor of javax.jcr.Node. It provides a new single method shown below:

```
/**
 * Returns a NodeIterator over all child Nodes of this Node. Does not include properties
 * of this Node. If this node has no child nodes, then an empty iterator is returned.
 *
 * @return A NodeIterator over all child Nodes of this <code>Node</code>.
 * @throws RepositoryException If an error occurs.
 */
public NodeIterator getNodesLazily() throws RepositoryException;
```

From the view of end-user or client application, getNodesLazily() works similar to JCR specified getNodes() returning NodeIterator. "Lazy" iterator supports the same set of features as an ordinary NodeIterator, including skip() and excluding remove() features. "Lazy" implementation performs reading from DB by pages. Each time when it has no more elements stored in memory, it reads next set of items from persistent layer. This set is called "page". Must admit that getNodesLazily feature fully supports session and transaction changes log, so it's a functionally-full analogue of specified getNodes() operation. So when having a deal with huge list of child nodes, getNodes() can be simply and safely substituted with getNodesLazily().

JCR gives an experimental opportunity to replace all getNodes() invocations with getNodesLazily() calls. It handles a boolean system property named "org.exoplatform.jcr.forceUserGetNodesLazily" that internally replaces one call with another, without any code changes. But be sure using it only for development purposes. This feature can be used with top level products using eXo JCR to perform a quick compatibility and performance tests without changing any code. This is not recommended to be used as a production solution.

Configuration

"Lazy" iterator makes ahead reading into memory, though reading the "page". "Page" is a set of nodes read at once. Size of the page by default is 100 nodes and can be configured though workspace container configuration using "lazy-node-iterator-page-size" parameter. I.e.:

```
<container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.CQJDBCWorkspaceDataContainer">
  <properties>
    <property name="source-name" value="jdbcjcr" />
    <property name="multi-db" value="true" />
    <property name="max-buffer-size" value="200k" />
    <property name="swap-directory" value="target/temp/swap/ws" />
    <property name="lazy-node-iterator-page-size" value="50" />
    ...
  </properties>
```

But it's not recommended to set huge page size

Implementation notices

Current "lazy" child nodes iterator supports caching, when pages are cached atomically in safe and optimized way. Cache is always kept in consistent state using invalidation if child list changed. Take in account the following difference in getNodes and getNodesLazily. Specification defined getNodes method reads whole list of nodes, so child items added after invocation will never be in results. GetNodesLazily doesn't acquire full list of nodes, so child items added after iterator creation can be found in result. So getNodesLazily can represent some kind of "real-time" results. But it is highly depend on numerous conditions and should not be used as a feature, it more likely implementation specific issue typical for "lazy-pattern".

WebDAV

Related documents

- *Link Producer*

Introduction

The WebDAV protocol enables you to use the third party tools to communicate with hierarchical content servers via HTTP. It is possible to add and remove documents or a set of documents from a path on the server. DeltaV is an extension of the WebDav protocol that allows managing document versioning. Locking guarantees protection against multiple access when writing resources. The ordering support allows changing the position of the resource in the list and sort the directory to make the directory tree viewed conveniently. The full-text search makes it easy to find the necessary documents. You can search by using two languages: SQL and XPATH.

In eXo JCR, we plug in the WebDAV layer - based on the code taken from the extension modules of the reference implementation - on the top of our JCR implementation so that it is possible to browse a workspace using the third party tools (it can be Windows folders or Mac ones as well as a Java WebDAV client, such as DAVExplorer or IE using File->Open as a Web Folder).

Now WebDav is an extension of the REST service. To get the WebDav server ready, you must deploy the REST application. Then, you can access any workspaces of your repository by using the following URL:

Standalone mode:

`http://host:port/rest/jcr/{RepositoryName}/{WorkspaceName}/{Path}`

Portal mode:

`http://host:port/portal/rest/private/jcr/{RepositoryName}/{WorkspaceName}/{Path}`

When accessing the WebDAV server with the URL`http://localhost:8080/rest/jcr/repository/production`, you might also use "collaboration" (instead of "production") which is the default workspace in eXo products. You will be asked to enter your login and password. Those will then be checked by using the organization service that can be implemented thanks to an InMemory (dummy) module or a DB module or an LDAP one and the JCR user session will be created with the correct JCR Credentials.

Note

If you try the "in ECM" option, add "@ecm" to the user's password. Alternatively, you may modify jaas.conf by adding the **domain=ecm** option as follows:

```
exo-domain {
```

```
      org.exoplatform.services.security.jaas.BasicLoginModule required domain=ecm;
  };
```

Configuration

```
<component>
  <key>org.exoplatform.services.webdav.WebDavServiceImpl</key>
  <type>org.exoplatform.services.webdav.WebDavServiceImpl</type>
  <init-params>

    <!-- this parameter indicates the default login and password values
         used as credentials for accessing the repository -->
    <!-- value-param>
      <name>default-identity</name>
      <value>admin:admin</value>
    </value-param -->

    <!-- this is the value of WWW-Authenticate header -->
    <value-param>
      <name>auth-header</name>
      <value>Basic realm="eXo-Platform Webdav Server 1.6.1"</value>
    </value-param>

    <!-- default node type which is used for the creation of collections -->
    <value-param>
      <name>def-folder-node-type</name>
      <value>nt:folder</value>
    </value-param>

    <!-- default node type which is used for the creation of files -->
    <value-param>
      <name>def-file-node-type</name>
      <value>nt:file</value>
    </value-param>

    <!-- if MimeTypeResolver can't find the required mime type,
         which conforms with the file extension, and the mimeType header is absent
         in the HTTP request header, this parameter is used
         as the default mime type-->
    <value-param>
      <name>def-file-mimetype</name>
      <value>application/octet-stream</value>
```

```
  </value-param>

 <!-- This parameter indicates one of the three cases when you update the content of the resource by PUT command.
       In case of "create-version", PUT command creates the new version of the resource if this resource exists.
        In case of "replace" - if the resource exists, PUT command updates the content of the resource and its last modification date.
       In case of "add", the PUT command tries to create the new resource with the same name (if the parent node allows same-name siblings).-->

  <value-param>
    <name>update-policy</name>
    <value>create-version</value>
    <!--value>replace</value -->
    <!-- value>add</value -->
  </value-param>

  <!--
      This parameter determines how service responds to a method that attempts to modify file content.
      In case of "checkout-checkin" value, when a modification request is applied to a checked-in version-controlled resource, the request is automatically preceded by a checkout and followed by a checkin operation.
      In case of "checkout" value, when a modification request is applied to a checked-in version-controlled resource, the request is automatically preceded by a checkout operation.
  -->
  <value-param>
    <name>auto-version</name>
    <value>checkout-checkin</value>
    <!--value>checkout</value -->
  </value-param>

  <!--
       This parameter is responsible for managing Cache-Control header value which will be returned to the client.
      You can use patterns like "text/*", "image/*" or wildcard to define the type of content.
  -->
  <value-param>
    <name>cache-control</name>
           <value>text/xml,text/html:max-age=3600;image/png,image/jpg:max-age=1800;*/*:no-cache;</value>
  </value-param>
```

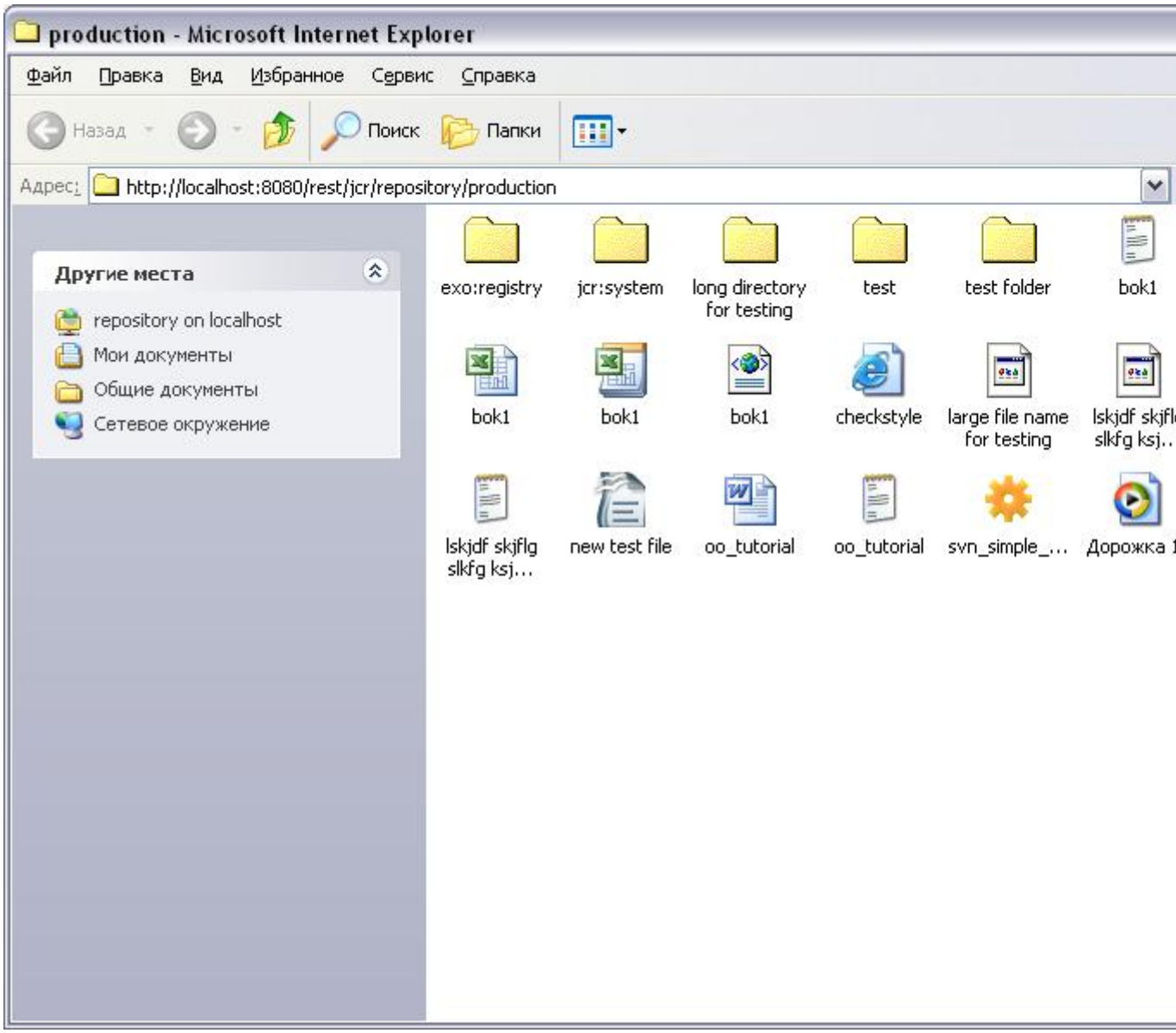
```
    <!--
       This parameter determines the absolute path to the folder icon file, which is shown
       during WebDAV view of the contents
    -->
    <value-param>
      <name>folder-icon-path</name>
      <value>/absolute/path/to/file</value>
    </value-param>

  </init-params
</component>
```

Screenshots

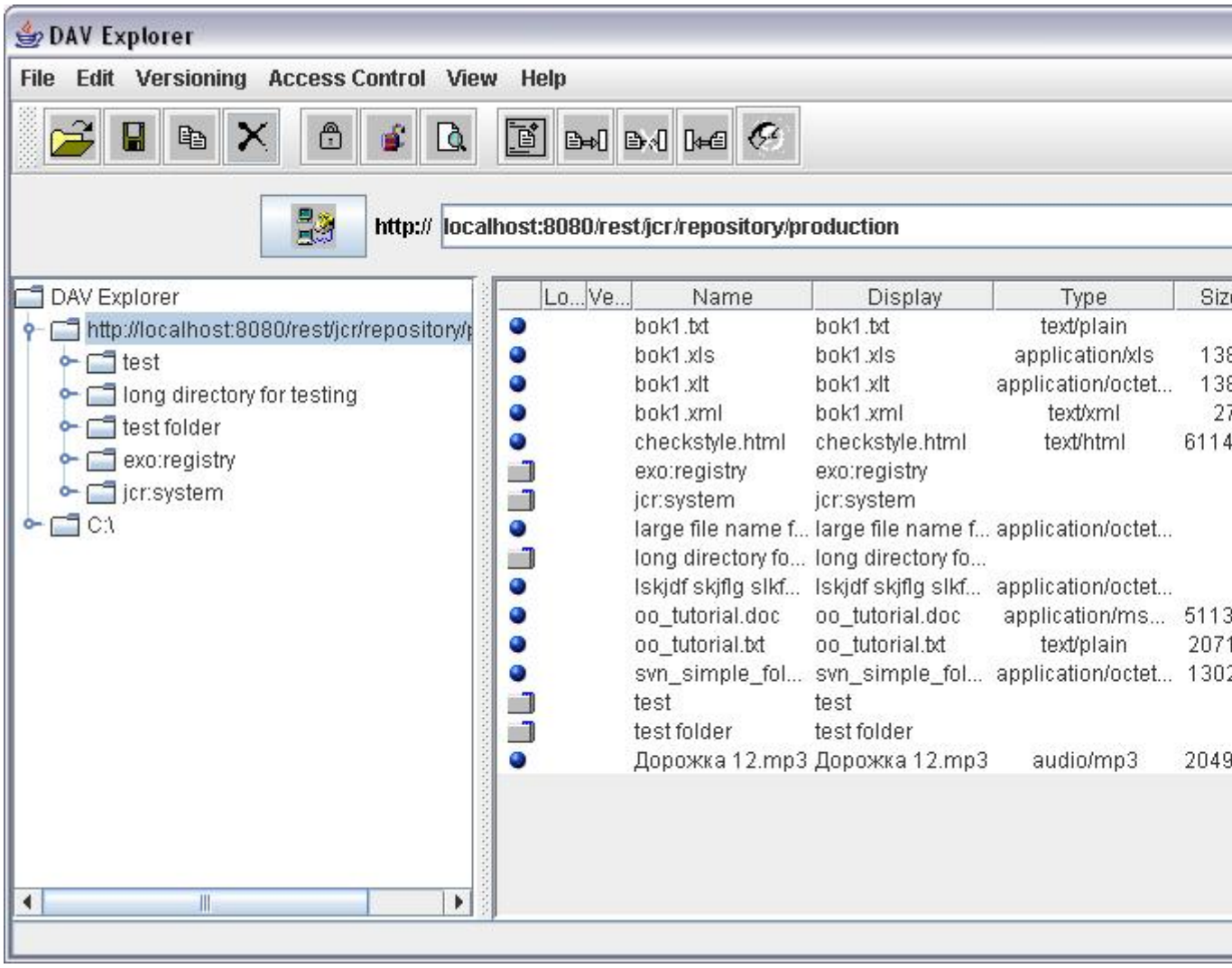
At present, eXo JCR WebDav server is tested by using MS Internet Explorer, *Dav Explorer* [http://www.ics.uci.edu/~webdav], *Xythos Drive* [http://www.xythos.com/home/xythos/products/xythos_drive.html], Microsoft Office 2003 (as client), and Ubuntu Linux.

MS Internet Explorer

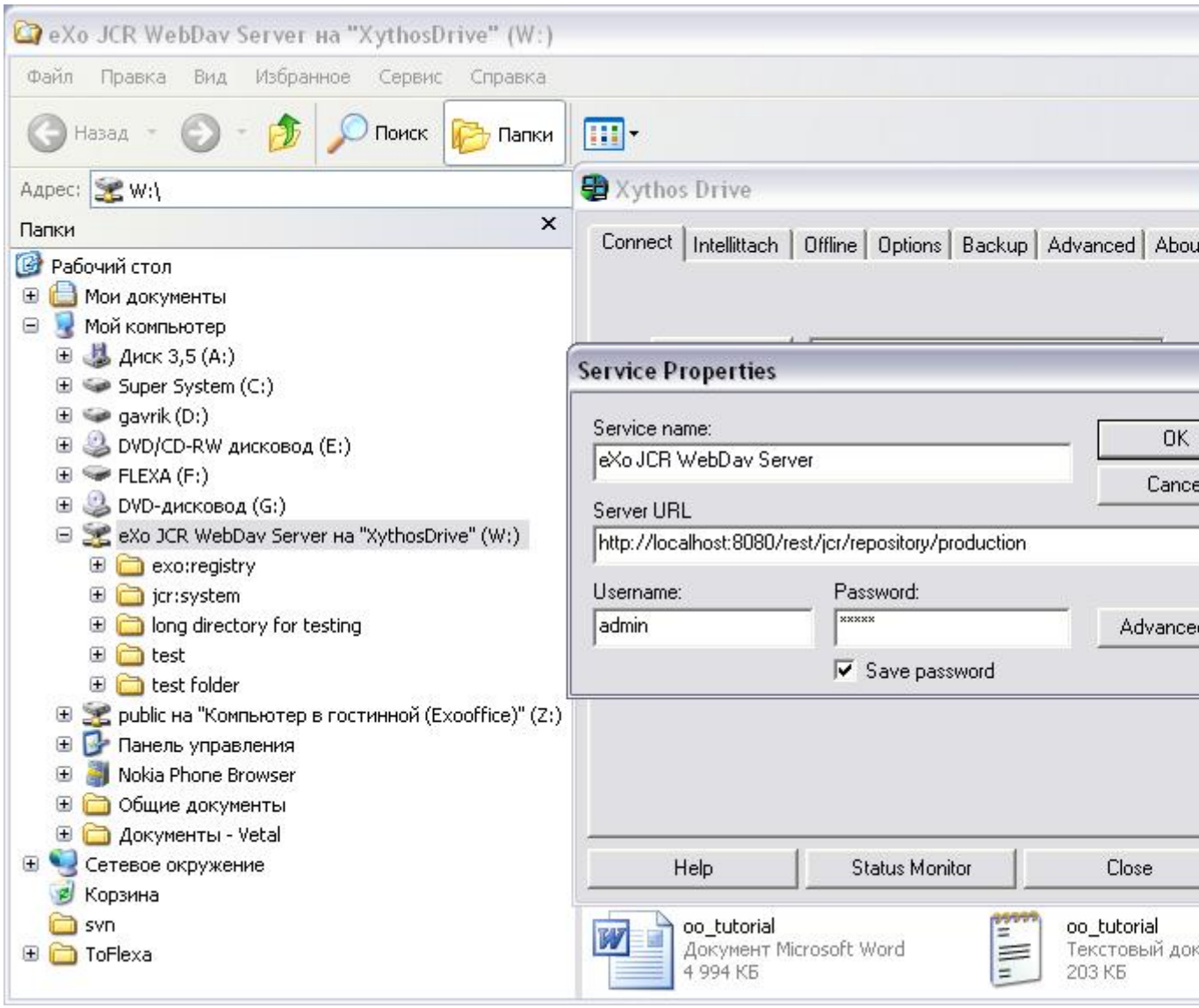
(File -> Open as Web Folder)


production - Microsoft Internet Explorer
Файл Правка Вид Избранное Сервис Справка
Назад
Поиск
Папки
Адрес: http://localhost:8080/rest/jcr/repository/production
Другие места
repository on localhost
Мои документы
Общие документы
Сетевое окружение
exo:registry
jcr:system
long directory for testing
test
test folder
bok1
bok1
bok1
bok1
checkstyle
large file name for testing
lskjdf skjflg slkfg ksj...
new test file
oo_tutorial
oo_tutorial
svn_simple_...

Dav Explorer

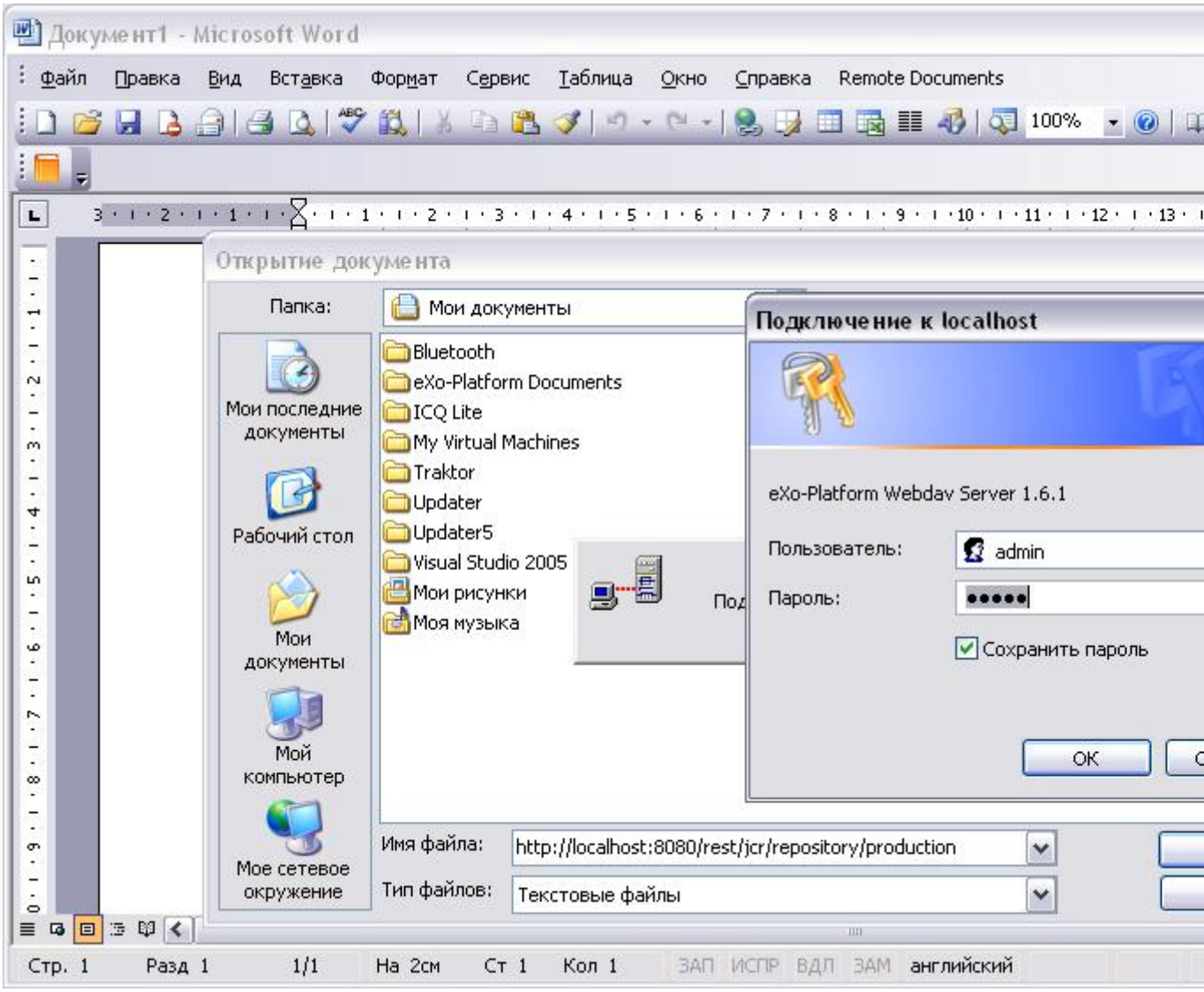


Xythos Drive

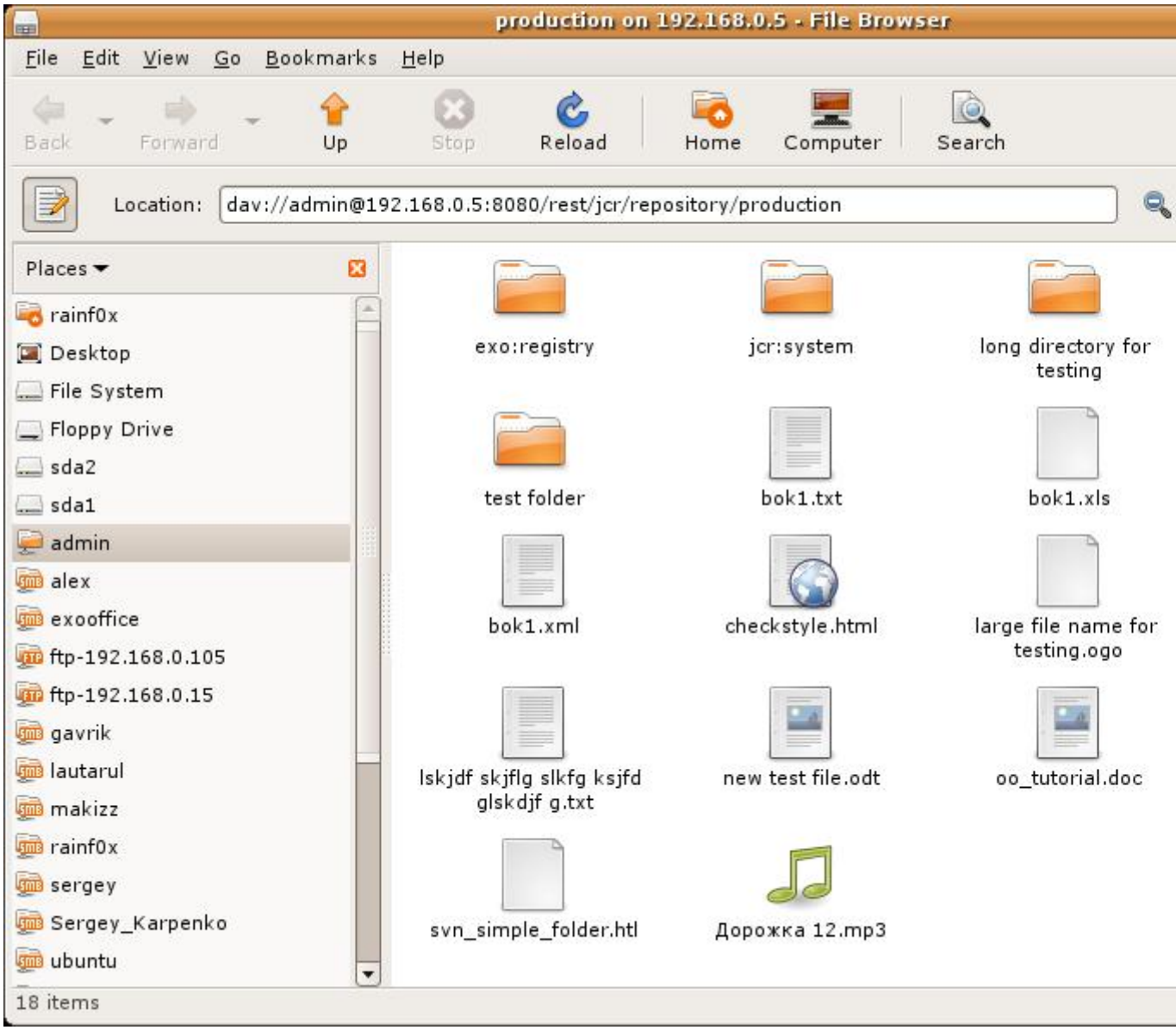


Microsoft Office 2003

(as client) (File->Open with typing http://... href in the file name box)


Документ1 - Microsoft Word
Файл Правка Вид Вставка Формат Сервис Таблица Окно Справка Remote Documents
100%
Открытие документа
Папка:
Мои документы
Мои последние документы
Рабочий стол
Мои документы
Мой компьютер
Мое сетевое окружение
Bluetooth
eXo-Platform Documents
ICQ Lite
My Virtual Machines
Traktor
Updater
Updater5
Visual Studio 2005
Мои рисунки
Моя музыка
Подключение к localhost
eXo-Platform Webdav Server 1.6.1
Пользователь:
admin
Пароль:
Сохранить пароль
OK
Имя файла:
http://localhost:8080/rest/jcr/repository/production
Тип файлов:
Текстовые файлы
Стр. 1 Разд 1 1/1 На 2см Ст 1 Кол 1 ЗАП ИСПР ВДЛ ЗАМ английский

Ubuntu Linux



Comparison table of WebDav and JCR commands

Table 32.1.

WebDav	JCR
COPY	Workspace.copy(...)
DELETE	Node.remove()
GET	Node.getProperty(...); Property.getValue()
HEAD	Node.getProperty(...); Property.getLength()
MKCOL	Node.addNode(...)
MOVE	Session.move(...) or Workspace.move(...)

WebDav	JCR
PROPFIND	Session.getNode(...); Node.getNode(...); Node.getNodes(...); Node.getProperties()
PROPPATCH	Node.setProperty(...); Node.getProperty(...).remove()
PUT	Node.addNode("node","nt:file"); Node.setProperty("jcr:data", "data")
CHECKIN	Node.checkin()
CHECKOUT	Node.checkout()
REPORT	Node.getVersionHistory(); VersionHistory.getAllVersions(); Version.getProperties()
RESTORE	Node.restore(...)
UNCHECKOUT	Node.restore(...)
VERSION-CONTROL	Node.addMixin("mix:versionable")
LOCK	Node.lock(...)
UNLOCK	Node.unlock()
ORDERPATCH	Node.orderBefore(...)
SEARCH	Workspace.getQueryManager(); QueryManager.createQuery(); Query.execute()

Restrictions

There are some restrictions for WebDAV in different Operating systems.

Windows 7

When you try to set up a web folder by “adding a network location” or “map a network drive” through My Computer, you can get an error message saying that either “The folder you entered does not appear to be valid. Please choose another” or “Windows cannot access… Check the spelling of the name. Otherwise, there might be…”. These errors may appear when you are using SSL or non-SSL.

To fix this, do as follows:

1. Go to Windows Registry Editor.
2. Find a key: \HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlset\services\WebClient\Parameters\BasicAuthLevel.

3. Change the value to 2.

Microsoft Office 2010

If you have Microsoft Office 2010 applications or Microsoft Office 2007 applications installed on a client computer. From that client computer, you try to access an Office file that is stored on a web server that is configured for Basic authentication. The connection between your computer and the web server does not use Secure Sockets Layer (SSL). When you try to open or to download the file, you experience the following symptoms:

- The Office file does not open or download.
- You do not receive a Basic authentication password prompt when you try to open or to download the file.
- You do not receive an error message when you try to open the file. The associated Office application starts. However, the selected file does not open.

To enable Basic authentication on the client computer, follow these steps:

1. Click Start, type regedit in the Start Search box, and then press Enter.
2. Locate and then click the following registry subkey:

   HKEY_CURRENT_USER\Software\Microsoft\Office\14.0\Common\Internet
3. On the Edit menu, point to New, and then click DWORD Value.
4. Type BasicAuthLevel, and then press Enter.
5. Right-click BasicAuthLevel, and then click Modify.
6. In the Value data box, type 2, and then click OK.

FTP

Introdution

The JCR-FTP Server represents the standard eXo service, operates as an FTP server with an access to a content stored in JCR repositories in the form of **nt:file/nt:folder** nodes or their successors. The client of an executed Server can be any FTP client. The FTP server is supported by a standard configuration which can be changed as required.

Configuration Parameters

command-port:

```
<value-param>
  <name>command-port</name>
  <value>21</value>
</value-param>
```

The value of the command channel port. The value '21' is default.

When you have already some FTP server installed in your system , this parameter needs to be changed (2121 for example) to avoid conflicts or if the port is protected.

data-min-port & data-max-port

```
<value-param>
  <name>data-min-port</name>
  <value>52000</value>
</value-param>
```

```
<value-param>
  <name>data-max-port</name>
  <value>53000</value>
</value-param>
```

These two parameters indicate the minimal and maximal values of the range of ports, used by the server. The usage of the additional data channel is required by the FTP - protocol, which is used to transfer the contents of files and the listing of catalogues. This range of ports should be free from listening by other server-programs.

system

```
<value-param>
  <name>system</name>

  <value>Windows_NT</value>
    or
  <value>UNIX Type: L8</value>
</value-param>
```

Types of formats of listing of catalogues which are supported.

client-side-encoding

```
<value-param>
  <name>client-side-encoding</name>

  <value>windows-1251</value>
    or
  <value>KOI8-R</value>

</value-param>
```

This parameter specifies the coding which is used for dialogue with the client.

def-folder-node-type

```
<value-param>
  <name>def-folder-node-type</name>
  <value>nt:folder</value>
</value-param>
```

This parameter specifies the type of a node, when an FTP-folder is created.

def-file-node-type

```
<value-param>
  <name>def-file-node-type</name>
  <value>nt:file</value>
```

```
</value-param>
```

This parameter specifies the type of a node, when an FTP - file is created.

def-file-mime-type

```
<value-param>
  <name>def-file-mime-type</name>
  <value>application/zip</value>
</value-param>
```

The mime type of a created file is chosen by using its file extention. In case, a server cannot find the corresponding mime type, this value is used.

cache-folder-name

```
<value-param>
  <name>cache-folder-name</name>
  <value>../temp/ftp_cache</value>
</value-param>
```

The Path of the cache folder.

upload-speed-limit

```
<value-param>
  <name>upload-speed-limit</name>
  <value>20480</value>
</value-param>
```

Restriction of the upload speed. It is measured in bytes.

download-speed-limit

```
<value-param>
  <name>download-speed-limit</name>
  <value>20480</value>
</value-param>
```

Restriction of the download speed. It is measured in bytes.

timeout

```
<value-param>
  <name>timeout</name>
  <value>60</value>
</value-param>
```

Defines the value of a timeout.

eXo JCR Backup Service

Note

Restore of system workspace is not supported only as part of restoring of whole repository.

Concept

The main purpose of that feature is to restore data in case of system faults and repository crashes. Also, the backup results may be used as a content history.

The concept is based on the export of a workspace unit in the Full, or Full + Incrementals model. A repository workspace can be backup and restored using a combination of these modes. In all cases, at least one Full (initial) backup must be executed to mark a starting point of the backup history. An Incremental backup is not a complete image of the workspace. It contains only changes for some period. So it is not possible to perform an Incremental backup without an initial Full backup.

The Backup service may operate as a hot-backup process at runtime on an in-use workspace. It's a case when the Full + Incrementals model should be used to have a guaranty of data consistency during restoration. An Incremental will be run starting from the start point of the Full backup and will contain changes that have occured during the Full backup, too.

A **restore** operation is a mirror of a backup one. At least one Full backup should be restored to obtain a workspace corresponding to some points in time. On the other hand, Incrementals may be restored in the order of creation to reach a required state of a content. If the Incremental contains the same data as the Full backup (hot-backup), the changes will be applied again as if they were made in a normal way via API calls.

According to the model there are several modes for backup logic:

- **Full backup only** : Single operation, runs once
- **Full + Incrementals** : Start with an initial Full backup and then keep incrementals changes in one file. Run until it is stopped.
- **Full + Incrementals(periodic)** : Start with an initial Full backup and then keep incrementals with periodic result file rotation. Run until it is stopped.

How it works

Implementation details

Full backup/restore is implemented using the JCR SysView Export/Import. Workspace data will be exported into Sysview XML data from root node.

Restoring is implemented, using the special eXo JCR API feature: a dynamic workspace creation. Restoring of the workspace Full backup will create one new workspace in the repository. Then, the SysView XML data will be imported as the root node.

Incremental backup is implemented using the eXo JCR ChangesLog API. This API allows to record each JCR API call as atomic entries in a changelog. Hence, the Incremental backup uses a listener that collects these logs and stores them in a file.

Restoring an incremental backup consists in applying the collected set of ChangesLogs to a workspace in the correct order.

Note

Incremental backup is an experimental feture and not supported, so it must be used with a lot of caution.

Work basics

The work of Backup is based on the BackupConfig configuration and the BackupChain logical unit.

BackupConfig describes the backup operation chain that will be performed by the service. When you intend to work with it, the configuration should be prepared before the backup is started.

The configuration contains such values as:

- **Types of full and incremental backup** (fullBackupType, incrementalBackupType): Strings with full names of classes which will cover the type functional.
- **Incremental period**: A period after that a current backup will be stopped and a new one will be started in seconds (long).
- **Target repository and workspace names**: Strings with described names
- **Destination directory** for result files: String with a path to a folder where operation result files will be stored.

BackupChain is a unit performing the backup process and it covers the principle of initial Full backup execution and manages Incrementals operations. BackupChain is used as a key object for accessing current backups during runtime via BackupManager. Each BackupJob performs a single atomic operation - a Full or Incremental process. The result of that operation is data for a Restore. BackupChain can contain one or more BackupJobs. But at least the initial Full job is always there. Each BackupJobs has its own unique number which means its Job order in the chain, the initial Full job always has the number 0.

Backup process, result data and file location

To start the backup process, it's necessary to create the BackupConfig and call the BackupManager.startBackup(BackupConfig) method. This method will return BackupChain created according to the configuration. At the same time, the chain creates a BackupChainLog

which persists BackupConfig content and BackupChain operation states to the file in the service working directory (see Configuration).

When the chain starts the work and the initial BackupJob starts, the job will create a result data file using the destination directory path from BackupConfig. The destination directory will contain a directory with an automatically created name using the pattern repository_workspace-timestamp where timestamp is current time in the format of yyyyMMdd_hhmmss (E.g. db1_ws1-20080306_055404). The directory will contain the results of all Jobs configured for execution. Each Job stores the backup result in its own file with the name repository_workspace-timestamp.jobNumber. BackupChain saves each state (STARTING, WAITING, WORKING, FINISHED) of its Jobs in the BackupChainLog, which has a current result full file path.

BackupChain log file and job result files are a whole and consistent unit, that is a source for a Restore.

Note

BackupChain log contains absolute paths to job result files. Don't move these files to another location.

Restore requirements

As mentioned before a Restore operation is a mirror of a Backup. The process is a Full restore of a root node with restoring an additional Incremental backup to reach a desired workspace state. Restoring of the workspace Full backup will create a new workspace in the repository using given RepositoyEntry of existing repository and given (preconfigured) WorkspaceEntry for a new target workspace. A Restore process will restore a root node from the SysView XML data.

Note

The target workspace should not be in the repository. Otherwise, a BackupConfigurationException exception will be thrown.

Finally, we may say that Restore is a process of a new Workspace creation and filling it with a Backup content. In case you already have a target Workspace (with the same name) in a Repository, you have to configure a new name for it. If no target workspace exists in the Repositor, you may use the same name as the Backup one.

Configuration

As an optional extension, the Backup service is not enabled by default. **You need to enable it via configuration**.

The following is an example configuration :

```
<component>
  <key>org.exoplatform.services.jcr.ext.backup.BackupManager</key>
```

```
  <type>org.exoplatform.services.jcr.ext.backup.impl.BackupManagerImpl</type>
  <init-params>
    <properties-param>
      <name>backup-properties</name>
      <property name="backup-dir" value="target/backup" />
    </properties-param>
  </init-params>
</component>
```

Where mandatory paramet is:

- **backup-dir** : The path to a working directory where the service will store internal files and chain logs.

Also, there are optional parameters:

- **incremental-backup-type** : The FQN of incremental job class. Must implement org.exoplatform.services.jcr.ext.backup.BackupJob. By default : org.exoplatform.services.jcr.ext.backup.impl.fs.FullBackupJob used.
- **default-incremental-job-period** : The period between incremetal flushes (in seconds). Default is 3600 seconds.
- **full-backup-type** : The FQN of the full backup job class; Must implement org.exoplatform.services.jcr.ext.backup.BackupJob. By default : org.exoplatform.services.jcr.ext.backup.impl.rdbms.FullBackupJob used. Please, notice that file-system based implementation org.exoplatform.services.jcr.ext.backup.impl.fs.FullBackupJob is deprecated and not recommended for use.

RDBMS backup

RDBMS backup It is the lastest, currently supportedm used by default and recommended implementation of full backup job for BackupManager service. It is useful in case when database is used to store data.

Brings such advantages:

- fast: backup takes only several minutes to perform full backup of repository with 1 million rows in tables;
- atomic restore: restore process into existing workspace/repository with same configuration is atomic, it means you don’t loose the data when restore failed, the original data remains;
- cluster aware: it is possible to make backup/restore in cluster environment into existing workspace/repository with same configuration;

- consistence backup: all threads make waiting until backup is finished and then continue to work, so, there are no data modification during backup process;

Usage

Performing a Backup

In the following example, we create a BackupConfig bean for the Full + Incrementals mode, then we ask the BackupManager to start the backup process.

```
// Obtaining the backup service from the eXo container.
BackupManager backup = (BackupManager) container.getComponentInstanceOfType(BackupManager.class);

// And prepare the BackupConfig instance with custom parameters.
// full backup & incremental
File backDir = new File("/backup/ws1"); // the destination path for result files
backDir.mkdirs();

BackupConfig config = new BackupConfig();
config.setRepository(repository.getName());
config.setWorkspace("ws1");
config.setBackupDir(backDir);

// Before 1.9.3, you also need to indicate the backupjobs class FDNs
// config.setFullBackupType("org.exoplatform.services.jcr.ext.backup.impl.fs.FullBackupJob");
//
config.setIncrementalBackupType("org.exoplatform.services.jcr.ext.backup.impl.fs.IncrementalBackupJob");

// start backup using the service manager
BackupChain chain = backup.startBackup(config);
```

To stop the backup operation, you have to use the BackupChain instance.

```
// stop backup
backup.stopBackup(chain);
```

Performing a Restore

Restoration involves reloading the backup file into a BackupChainLog and applying appropriate workspace initialization. The following snippet shows the typical sequence for restoring a workspace :

```
// find BackupChain using the repository and workspace names (return null if not found)
BackupChain chain = backup.findBackup("db1", "ws1");

// Get the RepositoryEntry and WorkspaceEntry
ManageableRepository repo = repositoryService.getRepository(repository);
RepositoryEntry repoconf = repo.getConfiguration();
List<WorkspaceEntry> entries = repoconf.getWorkspaceEntries();
WorkspaceEntry = getNewEntry(entries, workspace); // create a copy entry from an existing one

// restore backup log using ready RepositoryEntry and WorkspaceEntry
File backLog = new File(chain.getLogFilePath());
BackupChainLog bchLog = new BackupChainLog(backLog);

// initialize the workspace
repository.configWorkspace(workspaceEntry);

// run restoration
backup.restore(bchLog, repositoryEntry, workspaceEntry);
```

Restoring into an existing workspace

Note

These instructions only applies to regular workspace. Special instructions are provided for System workspace below.

To restore a backup over an existing workspace, you are required to clear its data. Your backup process should follow these steps:

- Remove workspace

```
ManageableRepository repo = repositoryService.getRepository(repository);
repo.removeWorkspace(workspace);
```

- Clean database, value storage, index
- Restore (see snippet above)

System workspace

Note

The BackupWorkspaceInitializer is available in JCR 1.9 and later.

Restoring the JCR System workspace requires to shutdown the system and use of a special initializer.

Follow these steps (this will also work for normal workspaces):

- Stop repository (or portal)
- Clean database, value storage, index;
- In configuration, the workspace set BackupWorkspaceInitializer to refer to your backup.

For example:

```
<workspaces>
  <workspace name="production" ... >
                                                                    <container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
      ...
    </container>
    <initializer class="org.exoplatform.services.jcr.impl.core.BackupWorkspaceInitializer">
      <properties>
         <property name="restore-path" value="D:\java\exo-working\backup\repository_production-
20090527_030434"/>
      </properties>
   </initializer>
    ...
</workspace>
```

- Start repository (or portal).

Repository and Workspace initialization from backup

Repository and Workspace initialization from backup can use the BackupWorkspaceInitializer.

Will be configured BackupWorkspaceInitializer in configuration of workspace to restore the Workspace from backup over initializer.

Will be configured BackupWorkspaceInitializer in all configurations workspaces of the Repository to restore the Repository from backup over initializer.

Restoring the repository or workspace requires to shutdown the repository.

Follow these steps:

- Stop repository (will be skipped this step if repository or workace is not exists)

- Clean database, value storage, index; (will be skipped this step if repository or worksace is new)
- In configuration, the workspace/-s set BackupWorkspaceInitializer to refer to your backup.
- Start repository

Example of configuration initializer to restore workspace "backup" over BackupWorkspaceInitializer:

```
<workspaces>
  <workspace name="backup" ... >
                                                                    <container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
      ...
    </container>
    <initializer class="org.exoplatform.services.jcr.impl.core.BackupWorkspaceInitializer">
      <properties>
            <property name="restore-path" value="D:\java\exo-working\backup\repository_backup-
20110120_044734"/>
      </properties>
   </initializer>
    ...
</workspace>
```

Restore the Workspace over BackupWorksaceInitializer

Example of configuration initializer to resore the workspace "backup" over BackupWorkspaceInitializer:

- Stop repository (will be skipped this step if workspace is not exists)
- Clean database, value storage, index; (will be skipped this step if workspace is new)

In configuration, the workspace/-s set BackupWorkspaceInitializer to refer to your backup.

```
<workspaces>
  <workspace name="backup" ... >
                                                                    <container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
      ...
    </container>
    <initializer class="org.exoplatform.services.jcr.impl.core.BackupWorkspaceInitializer">
      <properties>
```

```
        <property name="restore-path" value="D:\java\exo-working\backup\repository_backup-20110120_044734"/>
      </properties>
   </initializer>
    ...
</workspace>
```

- Start repository

Restore the Repository over BackupWorksaceInitializer

Example of configuration initializers to restore the repository "repository" over BackupWorkspaceInitializer:

- Stop repository (will be skipped this step if repository is not exists)
- Clean database, value storage, index; (will be skipped this step if repository is new)

In configuration of repository will be configured initializers of workspace to refer to your backup.

For example:

```
...
<workspaces>
 <workspace name="system" ... >
                                                                                <container class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
  ...
 </container>
 <initializer class="org.exoplatform.services.jcr.impl.core.BackupWorkspaceInitializer">
  <properties>
        <property name="restore-path" value="D:\java\exo-working\backup\repository_system-20110120_052334"/>
  </properties>
 </initializer>
 ...
</workspace>

 <workspace name="collaboration" ... >
                                                                                <container class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
   ...
 </container>
 <initializer class="org.exoplatform.services.jcr.impl.core.BackupWorkspaceInitializer">
```

```
  <properties>
    <property name="restore-path" value="D:\java\exo-working\backup\repository_collaboration-
20110120_052341"/>
  </properties>
 </initializer>
 ...
</workspace>

<workspace name="backup" ... >
                                                                                <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
 ...
 </container>

 <initializer class="org.exoplatform.services.jcr.impl.core.BackupWorkspaceInitializer">
  <properties>
        <property  name="restore-path"  value="D:\java\exo-working\backup\repository_backup-
20110120_052417"/>
  </properties>
 </initializer>
 ...
 </workspace>
</workspaces>
```

- Start repository.

Scheduling (experimental)

The Backup service has an additional feature that can be useful for a production level backup implementation. When you need to organize a backup of a repository, it's necessary to have a tool which will be able to create and manage a cycle of Full and Incremental backups in periodic manner.

The service has internal BackupScheduler which can run a configurable cycle of BackupChains as if they have been executed by a user during some period of time. I.e. BackupScheduler is a user-like daemon which asks the BackupManager to start or stop backup operations.

For that purpose, BackupScheduler has the method.

BackupScheduler.schedule(backupConfig, startDate, stopDate, chainPeriod, incrementalPeriod)

where

- backupConfig: A ready configuration which will be given to the BackupManager.startBackup() method

- startDate: The date and time of the backup start
- stopDate: The date and time of the backup stop
- chainPeriod: A period after which a current BackupChain will be stopped and a new one will be started in seconds
- incrementalPeriod: If it is greater than 0, it will be used to override the same value in backupConfig.

```
// geting the scheduler from the BackupManager
   BackupScheduler scheduler = backup.getScheduler();

// schedule backup using a ready configuration (Full + Incrementals) to run from startTime
// to stopTime. Full backuop will be performed every 24 hours (BackupChain lifecycle),
// incremental will rotate result files every 3 hours.
   scheduler.schedule(config, startTime, stopTime, 3600  * 24, 3600 * 3);

// it's possible to run the scheduler for an uncertain period of time (i.e. without stop time).
// schedule backup to run from startTime till it will be stopped manually
// also there, the incremental will rotate result files as it configured in BackupConfig
   scheduler.schedule(config, startTime, null, 3600 * 24, 0);

// to unschedule backup simply call the scheduler with the configuration describing the
// already planned backup cycle.
// the scheduler will search in internal tasks list for task with repository and
// workspace name from the configuration and will stop that task.
   scheduler.unschedule(config);
```

When the BackupScheduler starts the scheduling, it uses the internal Timer with startDate for the first (or just once) execution. If chainPeriod is greater than 0, then the task is repeated with this value used as a period starting from startDate. Otherwise, the task will be executed once at startDate time. If the scheduler has stopDate, it will stop the task (the chain cycle) after stopDate. And the last parameter incrementalPeriod will be used instead of the same from BackupConfig if its values are greater than 0.

Starting each task (BackupScheduler.schedule(...)), the scheduler creates a task file in the service working directory (see **Configuration**, backup-dir) which describes the task backup configuration and periodic values. These files will be used at the backup service start (JVM start) to reinitialize BackupScheduler for continuous task scheduling. Only tasks that don't have a stopDate or a stopDate not expired will be reinitialized.

There is one notice about BackupScheduler task reinitialization in the current implementation. It comes from the BackupScheduler nature and its implemented behaviour. As the scheduler is just a virtual user which asks the BackupManager to start or stop backup operations, it isn't able to

reinitialize each existing BackupChain before the service (JVM) is stopped. But it's possible to start a new operation with the same configuration via BackupManager (that was configured before and stored in a task file).

This is a main detail of the BackupScheduler which should be taken into suggestion of a backup operation design now. In case of reinitialization, the task will have new time values for the backup operation cycle as the chainPeriod and incrementalPeriod will be applied again. That behaviour may be changed in the future.

Restore existing workspace or repository

The resore of existing workspace or repositry is available.

For restore will be used spacial methods:

```
/**
   * Restore existing workspace. Previous data will be deleted.
   * For getting status of workspace restore can use
   * BackupManager.getLastRestore(String repositoryName, String workspaceName) method
   *
   * @param workspaceBackupIdentifier
   *          backup identifier
   * @param workspaceEntry
   *          new workspace configuration
   * @param asynchronous
   *          if 'true' restore will be in asynchronous mode (i.e. in separated thread)
   * @throws BackupOperationException
   *           if backup operation exception occurred
   * @throws BackupConfigurationException
   *           if configuration exception occurred
   */
     void restoreExistingWorkspace(String workspaceBackupIdentifier, String repositoryName,
 WorkspaceEntry workspaceEntry,
     boolean asynchronous) throws BackupOperationException, BackupConfigurationException;

  /**
   * Restore existing workspace. Previous data will be deleted.
   * For getting status of workspace restore use can use
   * BackupManager.getLastRestore(String repositoryName, String workspaceName) method
   *
   * @param log
   *          workspace backup log
   * @param workspaceEntry
   *          new workspace configuration
   * @param asynchronous
```

```
   *          if 'true' restore will be in asynchronous mode (i.e. in separated thread)
   * @throws BackupOperationException
   *           if backup operation exception occurred
   * @throws BackupConfigurationException
   *           if configuration exception occurred
   */
          void restoreExistingWorkspace(BackupChainLog log, String repositoryName,
WorkspaceEntry workspaceEntry, boolean asynchronous)  throws BackupOperationException,
BackupConfigurationException;

  /**
   * Restore existing repository. Previous data will be deleted.
   * For getting status of repository restore can use
   * BackupManager.getLastRestore(String repositoryName) method
   *
   * @param repositoryBackupIdentifier
   *          backup identifier
   * @param repositoryEntry
   *          new repository configuration
   * @param asynchronous
   *          if 'true' restore will be in asynchronous mode (i.e. in separated thread)
   * @throws BackupOperationException
   *           if backup operation exception occurred
   * @throws BackupConfigurationException
   *           if configuration exception occurred
   */
       void restoreExistingRepository(String repositoryBackupIdentifier, RepositoryEntry
    repositoryEntry, boolean asynchronous) throws BackupOperationException,
BackupConfigurationException;

  /**
   * Restore existing repository. Previous data will be deleted.
   * For getting status of repository restore can use
   * BackupManager.getLastRestore(String repositoryName) method
   *
   * @param log
   *          repository backup log
   * @param repositoryEntry
   *          new repository configuration
   * @param asynchronous
   *          if 'true' restore will be in asynchronous mode (i.e. in separated thread)
   * @throws BackupOperationException
   *           if backup operation exception occurred
   * @throws BackupConfigurationException
```

```
   *          if configuration exception occurred
   */
          void restoreExistingRepository(RepositoryBackupChainLog log, RepositoryEntry
repositoryEntry, boolean asynchronous)
     throws BackupOperationException, BackupConfigurationException;
```

These methods for restore will do:

- remove existed workspace or repository;
- clean database;
- clean index data;
- clean value storage;
- restore from backup.

Restore a workspace or a repository using original configuration

The Backup manager allows you to restore a repository or a workspace using the original configuration stored into the backup log:

```
/**
   * Restore existing workspace. Previous data will be deleted.
   * For getting status of workspace restore can use
   * BackupManager.getLastRestore(String repositoryName, String workspaceName) method
   * WorkspaceEntry for restore should be contains in BackupChainLog.
   *
   * @param workspaceBackupIdentifier
   *          identifier to workspace backup.
   * @param asynchronous
   *          if 'true' restore will be in asynchronous mode (i.e. in separated thread)
   * @throws BackupOperationException
   *           if backup operation exception occurred
   * @throws BackupConfigurationException
   *          if configuration exception occurred
   */
  void restoreExistingWorkspace(String workspaceBackupIdentifier, boolean asynchronous)
          throws BackupOperationException,
          BackupConfigurationException;

  /**
   * Restore existing repository. Previous data will be deleted.
```

```
   * For getting status of repository restore can use
   * BackupManager.getLastRestore(String repositoryName) method.
   * ReprositoryEntry for restore should be contains in BackupChainLog.
   *
   * @param repositoryBackupIdentifier
   *          identifier to repository backup.
   * @param asynchronous
   *          if 'true' restore will be in asynchronous mode (i.e. in separated thread)
   * @throws BackupOperationException
   *           if backup operation exception occurred
   * @throws BackupConfigurationException
   *           if configuration exception occurred
   */
  void restoreExistingRepository(String repositoryBackupIdentifier, boolean asynchronous)
          throws BackupOperationException,
          BackupConfigurationException;

  /**
   * WorkspaceEntry for restore should be contains in BackupChainLog.
   *
   * @param workspaceBackupIdentifier
   *          identifier to workspace backup.
   * @param asynchronous
   *          if 'true' restore will be in asynchronous mode (i.e. in separated thread)
   * @throws BackupOperationException
   *           if backup operation exception occurred
   * @throws BackupConfigurationException
   *           if configuration exception occurred
   */
    void restoreWorkspace(String workspaceBackupIdentifier, boolean asynchronous) throws
BackupOperationException,
          BackupConfigurationException;

  /**
   * ReprositoryEntry for restore should be contains in BackupChainLog.
   *
   * @param repositoryBackupIdentifier
   *          identifier to repository backup.
   * @param asynchronous
   *          if 'true' restore will be in asynchronous mode (i.e. in separated thread)
   * @throws BackupOperationException
   *           if backup operation exception occurred
   * @throws BackupConfigurationException
   *           if configuration exception occurred
```

```
   */
    void restoreRepository(String repositoryBackupIdentifier, boolean asynchronous) throws
 BackupOperationException,
          BackupConfigurationException;

   /**
   * Restore existing workspace. Previous data will be deleted.
   * For getting status of workspace restore can use
   * BackupManager.getLastRestore(String repositoryName, String workspaceName) method
   * WorkspaceEntry for restore should be contains in BackupChainLog.
   *
   * @param workspaceBackupSetDir
   *          the directory with backup set
   * @param asynchronous
   *          if 'true' restore will be in asynchronous mode (i.e. in separated thread)
   * @throws BackupOperationException
   *           if backup operation exception occurred
   * @throws BackupConfigurationException
   *           if configuration exception occurred
   */
  void restoreExistingWorkspace(File workspaceBackupSetDir, boolean asynchronous)
          throws BackupOperationException, BackupConfigurationException;

  /**
   * Restore existing repository. Previous data will be deleted.
   * For getting status of repository restore can use
   * BackupManager.getLastRestore(String repositoryName) method.
   * ReprositoryEntry for restore should be contains in BackupChainLog.
   *
   * @param repositoryBackupSetDir
   *          the directory with backup set
   * @param asynchronous
   *          if 'true' restore will be in asynchronous mode (i.e. in separated thread)
   * @throws BackupOperationException
   *           if backup operation exception occurred
   * @throws BackupConfigurationException
   *           if configuration exception occurred
   */
  void restoreExistingRepository(File repositoryBackupSetDir, boolean asynchronous)
          throws BackupOperationException, BackupConfigurationException;

  /**
   * WorkspaceEntry for restore should be contains in BackupChainLog.
   *
```

```
   * @param workspaceBackupSetDir
   *          the directory with backup set
   * @param asynchronous
   *          if 'true' restore will be in asynchronous mode (i.e. in separated thread)
   * @throws BackupOperationException
   *           if backup operation exception occurred
   * @throws BackupConfigurationException
   *           if configuration exception occurred
   */
      void restoreWorkspace(File workspaceBackupSetDir, boolean asynchronous) throws
BackupOperationException,
          BackupConfigurationException;

  /**
   * ReprositoryEntry for restore should be contains in BackupChainLog.
   *
   * @param repositoryBackupSetDir
   *          the directory with backup set
   * @param asynchronous
   *          if 'true' restore will be in asynchronous mode (i.e. in separated thread)
   * @throws BackupOperationException
   *           if backup operation exception occurred
   * @throws BackupConfigurationException
   *           if configuration exception occurred
   */
      void restoreRepository(File repositoryBackupSetDir, boolean asynchronous) throws
BackupOperationException,
          BackupConfigurationException;
```

Backup set portability

The Backup log is stored during the Backup operation into two different locations: backup-dir directory of BackupService to support interactive operations via Backup API (e.g. console) and backup set files for portability (e.g. on another server).

HTTPBackupAgent and backup client

> For this service, you should configure the **org.exoplatform.services.jcr.impl.config.JDBCConfigurationPersister** in order to save the changes of the repository configuration. See the *eXo JCR Configuration article at chapter '2 Portal and Standalone configuration'* .

GateIn uses context /portal/rest, therefore you need to use http://host:port/portal/rest/ instread of http://host:port/rest/

GateIn uses form authentication, so first you need to login (url to form authentication is http://host:port/portal/login) and then perform requests.

Introduction

The service org.exoplatform.services.jcr.ext.backup.server.HTTPBackupAgent is REST-based front-end to service org.exoplatform.services.jcr.ext.backup.BackupManager. HTTPBackupAgent is representation BackupManager to creation backup, restore, getting status of current or completed backup/restore, etc.

The backup client is http client for HTTPBackupAgent.

HTTPBackupAgent

The HTTPBackupAgent is based on REST (see details about the *REST Framework*).

HTTPBackupAgent is using POST and GET methods for request.

The HTTPBackupAgent allows :

- Start backup
- Stop backup
- Restore from backup
- Delete the workspace
- Get information about backup service (BackupManager)
- Get information about current backup / restores / completed backups

HTTPBackupAgent methods

Starting Backup Service

/rest/jcr-backup/start/{repo}/{ws}

Start backup on specific workspace

URL: `http://host:port/rest/jcr-backup/start/{repo}/{ws}`

Formats: json.

Method: POST

Parameters:

- {repo} - the repository name;
- {ws} - the workspace name;
- BackupConfigBean - the JSON to BackupConfigBean.

The BackupConfigBean:

```
header :
"Content-Type" = "application/json; charset=UTF-8"

body:
<JSON to BackupConfigBean>
```

The JSON bean of org.exoplatform.services.jcr.ext.backup.server.bean.BackupConfigBean :

```
{"incrementalRepetitionNumber":<Integer>,"incrementalBackupJobConfig":<JSON to
 BackupJobConfig>,
"backupType":<Integer>,"fullBackupJobConfig":<JSON to BackupJobConfig>,
"incrementalJobPeriod":<Long>,"backupDir":"<String>"}
```

Where :

```
backupType                - the type of backup:
                          0 - full backup only;
                          1 - full and incremental backup.
backupDir                 - the path to backup folder;
incrementalJobPeriod      - the incremental job period;
incrementalRepetitionNumber - the incremental repetition number;
```

```
fullBackupJobConfig         - the configuration to full backup, JSON to BackupJobConfig;
incrementalJobPeriod        - the configuration to incremental backup, JSON to BackupJobConfig.
```

The JSON bean of org.exoplatform.services.jcr.ext.backup.server.bean.response.BackupJobConfig :

```
{"parameters":[<JSON to Pair>, ..., <JSON to pair> ],"backupJob":"<String>"}
```

Where:

```
backupJob  - the FQN (fully qualified name) to BackupJob class;
parameters - the list of JSON of Pair.
```

The JSON bean of org.exoplatform.services.jcr.ext.backup.server.bean.response.Pair :

```
{"name":"<String>","value":"<String>"}
```

Where:

```
name  - the name of parameter;
value - the value of parameter.
```

Returns:

- Return when being successful

```
status code = 200
```

- Return when being failure

```
status code = 404           - the not found repositry '{repo}' or workspace '{ws}'
status code = 500           - the other unknown errors
failure message in response  - the description of failure
```

Stopping Backup Service

/rest/jcr-backup/stop/{id}

Stop backup with identifier {id}.

URL: `http://host:port/rest/jcr-backup/stop/{id}`

Formats: plain text

Method: GET

Parameters:

- {id} - the identifier of backup

Returns:

- Return when being successful

```
status code = 200
```

- Return when being failure

```
status code = 404          - the no active backup with identifier {id}
status code = 500          - the other unknown errors
failure message in response  - the description of failure
```

Backup Info Service

/rest/jcr-backup/info

Information about the backup service.

URL: `http://host:port/rest/jcr-backup/info`

Formats: json

Method: GET

Parameters: no

Returns:

- Return when being successful

  Return the JSON bean of org.exoplatform.services.jcr.ext.backup.server.bean.response.BackupServiceInfoBean :

```
{"backupLogDir":"<String>","defaultIncrementalJobPeriod":<Long>,"fullBackupType":"<String>","incrementalBack
```

  Where:

```
fullBackupType           - the FQN (fully qualified name) of BackupJob class for full backup type;
incrementalBackupType    - the FQN (fully qualified name) of BackupJob class for incremental
 backup type;
backupLogDir             - path to backup folder;
defaultIncrementalJobPeriod - the default incremental job period.
```

- Return when being failure

```
status code = 500          - the unknown error
failure message in response  - the description of failure
```

Dropping Workspace Service

/rest/jcr-backup/drop-workspace/{repo}/{ws}/{force-session-close}

Delete the workspace from repository /{repo}/{ws}. With this service, you can delete any workspaces regardless of whether the workspace is a backup or has been copied to a backup.

URL: `http://host:port/rest/jcr-backup/drop-workspace/{repo}/{ws}/{force-session-close}`

Formats: plain text

Method: GET

Parameters:

- {repo} - the repository name;
- {ws} - the workspace name;
- {force-session-close} - the boolean value : **true** - the open sessions on workspace will be closed; **false** - will not close open sessions.

Returns:

- Return when being successful.

```
status code = 200
```

- Return when being failure

```
status code = 500          - the other unknown errors;
```

```
                        - not found repositry '{repo}' or workspace '{ws}'
failure message in response  - the description of failure
```

Backup Info

/rest/jcr-backup/info/backup

Information about the current and completed backups

URL: `http://host:port/rest/jcr-backup/info/backup`

Formats: json

Method: GET

Parameters: no

Returns:

- Return when being successful

  The JSON bean of org.exoplatform.services.jcr.ext.backup.server.bean.response.ShortInfoList :

```
{"backups":[<JSON to ShortInfo>,<JSON to ShortInfo>,...,<JSON to ShortInfo>]}
```

  The JSON bean of org.exoplatform.services.jcr.ext.backup.server.bean.response.ShortInfo :

```
{"startedTime":"<String>","backupId":"<String>","type":<Integer>,"state":<Integer>,"backupType":<Integer>,
"workspaceName":"<String>","finishedTime":"<String>","repositoryName":"<String>"}
```

  Where:

```
type           - the type of ShortInfo :
                  0 - the ShorInfo to completed backup;
                 -1 - the ShorInfo to current (active) backup.
                  1 - the ShorInfo to current restore.
backupType     - the type of backup:
                  0 - full backup only;
                  1 - full and incremental backup.
backupId       - the identifier of backup;
workspaceName  - the name of workspace;
repositoryName - the name of repository.
```

```
startedTime    - the date of started backup. The date in format RFC 1123 (for examle "Thu, 16
 Apr 2009 14:56:49 EEST").

The ShorInfo to current (active) backup :
  finishedTime - no applicable, always an empty string ("");
  state        - the state of full backup :
                 0 - starting;
                 1 - waiting;
                 2 - working;
                 4 - finished.

The ShorInfo to completed backup :
  finishedTime - the date of finished backup. The date in format RFC 1123;
  state        - no applicable, always zero (0).

The ShorInfo to current restore :
  finishedTime - the date of finished backup. The date in format RFC 1123;
  state        - the state of restore :
                 1 - started;
                 2 - successful;
                 3 - failure;
                 4 - initialized.
```

- Return when being failure

```
status code = 500            - the unknown error
failure message in response  - the description of failure
```

Current Backups Information

/rest/jcr-backup/info/backup/current Information about the current backups

URL: `http://host:port/rest/jcr-backup/info/backup/current`

Formats: json

Method: GET

Parameters: no

Returns:

- Return when being successful

  The JSON bean of org.exoplatform.services.jcr.ext.backup.server.bean.response.ShortInfoList (see item **/rest/jcr-backup/info/backup**)

- Return when being failure

```
status code = 500           - the unknown error
failure message in response  - the description of failure
```

Completed Backups Information

/rest/jcr-backup/info/backup/completed Information about the completed backups.

URL: `http://host:port/rest/jcr-backup/info/backup/completed`

Formats: json

Method: GET

Parameters: no

Returns:

- Return when being successful

  The JSON bean of org.exoplatform.services.jcr.ext.backup.server.bean.response.ShortInfoList (see item **/rest/jcr-backup/info/backup**)

- Return when being failure

```
status code = 500           - the unknown error
failure message in response  - the description of failure
```

Workspace-specific Backup Information

/rest/jcr-backup/info/backup/{repo}/{ws} Information about the current and completed backups for specific workspace.

URL: `http://host:port/rest/jcr-backup/info/backup/{repo}/{ws}`

Formats: json

Method: GET

Parameters:

- {repo} - the repository name
- {ws} - the workspace name

Returns:

- Return when being successful

The JSON bean of org.exoplatform.services.jcr.ext.backup.server.bean.response.ShortInfoList (see item **/rest/jcr-backup/info/backup**)

- Return when being failure

```
status code = 500          - the unknown error
failure message in response  - the description of failure
```

Single Backup Information

/rest/jcr-backup/info/backup/{id} Detailed information about a current or completed backup with identifier '{id}'.

URL: `http://host:port/rest/jcr-backup/info/backup/{id}`

Formats: json

Method: GET

Parameters:

- {id} - the identifier of backup

Returns:

- Return when being successful

  The JSON bean of org.exoplatform.services.jcr.ext.backup.server.bean.response.DetailedInfo :

```
{"backupConfig":<JSON                                                    to
 BackupConfigBean>,"startedTime":"<String>","backupId":"<String>","type":<Integer>,
"state":<Integer>,"backupType":<Integer>,"workspaceName":"<String>","finishedTime":"<String>",
"repositoryName":"<String>"}
```

Where:

```
type          - the type of DetailedInfo :
                 0 - the DetailedInfo to completed backup;
                -1 - the DetailedInfo to current (active) backup;
                 1 - the DetailedInfo to restore.
backupType     - the type of backup:
                 0 - full backup only;
                 1 - full and incremental backup.
backupId       - the identifier of backup;
```

```
workspaceName  - the name of workspace;
repositoryName - the name of repository;
backupConfig   - the JSON to BackupConfigBean.

The DetailedInfo to current (active) backup :
  startedTime  - the date of started backup. The date in format RFC 1123 (for examle "Thu, 16
 Apr 2009 14:56:49 EEST");
  finishedTime - no applicable, always an empty string ("");
  state        - the state of full backup :
                 0 - starting;
                 1 - waiting;
                 2 - working;
                 4 - finished.
The DetailedInfo to completed backup :
  startedTime  - the date of started backup. The date in format RFC 1123 (for examle "Thu, 16
 Apr 2009 14:56:49 EEST");
  finishedTime - the date of finished backup. The date in format RFC 1123;
  state        - no applicable, always zero (0).

The DetailedInfo to restore :
  startedTime  - the date of started restore. The date in format RFC 1123 (for examle "Thu, 16
 Apr 2009 14:56:49 EEST");
  finishedTime - the date of finished restore;
  state        - the state of restore :
                 1 - started;
                 2 - successful;
                 3 - failure;
                 4 - initialized.
```

The JSON bean of org.exoplatform.services.jcr.ext.backup.server.bean.BackupConfigBean (see item **/rest/jcr-backup/start/{repo}/{ws}**).

- Return when being failure

```
status code = 404          - not found the backup with {id}
status code = 500          - the unknown error
failure message in response  - the description of failure
```

Restores on a Workspace Information

/rest/jcr-backup/info/restore/{repo}/{ws} The information about the last restore on a specific workspace /{repo}/{ws}.

URL: `http://host:port/rest/jcr-backup/info/restore/{repo}/{ws}`

Formats: json

Method: GET

Parameters:

- {repo} - the repository name
- {ws} - the workspace name

Returns:

- Return when being successful

  The JSON bean of org.exoplatform.services.jcr.ext.backup.server.bean.response.DetailedInfo (see item **/rest/jcr-backup/info/backup/{id}**)

- Return when being failure

```
status code = 404          - the not found the restore for workspace /{repo}/{ws}
status code = 500          - the unknown error
failure message in response  - the description of failure
```

Restores Information

/rest/jcr-backup/info/restores

The information about the last restores.

URL: `http://host:port/rest/jcr-backup/info/restores`

Formats: json

Method: GET

Parameters: no

Returns:

- Return when being successful

  The JSON bean of org.exoplatform.services.jcr.ext.backup.server.bean.response.ShortInfoList (see item **/rest/jcr-backup/info/backup**)

- Return when being failure

```
status code = 500          - the unknown error
failure message in response  - the description of failure
```

Restoring Service

/rest/jcr-backup/restore/{repo}/{id}

Restore the workspace from specific backup.

URL: `http://host:port/rest/jcr-backup/restore/{repo}/{id}`

Formats: json.

Method: POST

Parameters:

- {repo} - the repository name;
- {id} - the identifier to backup; * WorkspaceEntry - the JSON to WorkspaceEntry.

The RestoreBean:

```
header :
"Content-Type" = "application/json; charset=UTF-8"

body:
<JSON to WorkspaceEntry>
```

The example of JSON bean to org.exoplatform.services.jcr.config.WorkspaceEntry :

```
{ "accessManager" : null,
  "autoInitPermissions" : null,
  "autoInitializedRootNt" : null,
  "cache" : { "parameters" : [ { "name" : "max-size",
            "value" : "10k"
          },
          { "name" : "live-time",
            "value" : "1h"
          }
        ],
                                                                    "type"                :
 "org.exoplatform.services.jcr.impl.dataflow.persistent.LinkedWorkspaceStorageCacheImpl"
    },
  "container" : { "parameters" : [ { "name" : "source-name",
            "value" : "jdbcjcr"
          },
          { "name" : "dialect",
            "value" : "hsqldb"
```

```
        },
        { "name" : "multi-db",
          "value" : "false"
        },
        { "name" : "update-storage",
          "value" : "false"
        },
        { "name" : "max-buffer-size",
          "value" : "200k"
        },
        { "name" : "swap-directory",
          "value" : "../temp/swap/production"
        }
      ],
    "type" : "org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer",
    "valueStorages" : [ { "filters" : [ { "ancestorPath" : null,
                "minValueSize" : 0,
                "propertyName" : null,
                "propertyType" : "Binary"
              } ],
          "id" : "system",
          "parameters" : [ { "name" : "path",
                "value" : "../temp/values/production"
              } ],
          "type" : "org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValueStorage"
        } ]
  },
"initializer" : { "parameters" : [ { "name" : "root-nodetype",
          "value" : "nt:unstructured"
        } ],
    "type" : "org.exoplatform.services.jcr.impl.core.ScratchWorkspaceInitializer"
  },
"lockManager" : { "persister" : { "parameters" : [ { "name" : "path",
              "value" : "../temp/lock/system"
            } ],
        "type" : "org.exoplatform.services.jcr.impl.core.lock.FileSystemLockPersister"
      },
    "timeout" : 15728640
  },
"name" : "production",
"queryHandler" : { "analyzer" : {  },
    "autoRepair" : true,
    "bufferSize" : 10,
    "cacheSize" : 1000,
```

```
    "documentOrder" : true,
    "errorLogSize" : 50,
                                                    "excerptProviderClass" :
"org.exoplatform.services.jcr.impl.core.query.lucene.DefaultHTMLExcerpt",
    "excludedNodeIdentifers" : null,
    "extractorBackLogSize" : 100,
    "extractorPoolSize" : 0,
    "extractorTimeout" : 100,
    "indexDir" : "../temp/jcrlucenedb/production",
                                                    "indexingConfigurationClass" :
"org.exoplatform.services.jcr.impl.core.query.lucene.IndexingConfigurationImpl",
    "indexingConfigurationPath" : null,
    "maxFieldLength" : 10000,
    "maxMergeDocs" : 2147483647,
    "mergeFactor" : 10,
    "minMergeDocs" : 100,
    "parameters" : [ { "name" : "index-dir",
        "value" : "../temp/jcrlucenedb/production"
      } ],
    "queryClass" : "org.exoplatform.services.jcr.impl.core.query.QueryImpl",
    "queryHandler" : null,
    "resultFetchSize" : 2147483647,
    "rootNodeIdentifer" : "00exo0jcr0root0uuid0000000000000",
    "spellCheckerClass" : null,
    "supportHighlighting" : false,
    "synonymProviderClass" : null,
    "synonymProviderConfigPath" : null,
    "type" : "org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex",
    "useCompoundFile" : false,
    "volatileIdleTime" : 3
  },
 "uniqueName" : "repository_production"
}
```


Returns:

- Return when being successful

```
status code = 200
```

Return the JSON bean org.exoplatform.services.jcr.ext.backup.server.bean.response.ShortInfo of just started restore. For JSON description see item **/rest/jcr-backup/info/backup**

- Return when being failure

```
status code = 403          - the already was restore to workspace /{repo}/{ws}
status code = 404          - the not found repositry '{repo}' or unsupported encoding to
 workspaceConfig
status code = 500          - the other unknown errors
failure message in response  - the description of failure
```

Default Workspace Information

/rest/jcr-backup/info/default-ws-config Will be returned the JSON bean to WorkspaceEntry for default workspace.

URL: `http://host:port/rest/jcr-backup/info/default-ws-config`

Formats: json

Method: GET

Parameters: no

Returns:

- Return when being successful

  The JSON bean to org.exoplatform.services.jcr.config.WorkspaceEntry :

```
{ "accessManager" : null,
  "autoInitPermissions" : null,
  "autoInitializedRootNt" : null,
  "cache" : { "parameters" : [ { "name" : "max-size",
            "value" : "10k"
          },
          { "name" : "live-time",
            "value" : "1h"
          }
        ],
                                                                    "type"          :
 "org.exoplatform.services.jcr.impl.dataflow.persistent.LinkedWorkspaceStorageCacheImpl"
    },
  "container" : { "parameters" : [ { "name" : "source-name",
            "value" : "jdbcjcr"
          },
          { "name" : "dialect",
            "value" : "hsqldb"
          },
```

```
        { "name" : "multi-db",
          "value" : "false"
        },
        { "name" : "update-storage",
          "value" : "false"
        },
        { "name" : "max-buffer-size",
          "value" : "200k"
        },
        { "name" : "swap-directory",
          "value" : "../temp/swap/production"
        }
      ],
    "type" : "org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer",
    "valueStorages" : [ { "filters" : [ { "ancestorPath" : null,
              "minValueSize" : 0,
              "propertyName" : null,
              "propertyType" : "Binary"
            } ],
        "id" : "system",
        "parameters" : [ { "name" : "path",
            "value" : "../temp/values/production"
          } ],
        "type" : "org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValueStorage"
      } ]
  },
"initializer" : { "parameters" : [ { "name" : "root-nodetype",
        "value" : "nt:unstructured"
      } ],
    "type" : "org.exoplatform.services.jcr.impl.core.ScratchWorkspaceInitializer"
  },
"lockManager" : { "persister" : { "parameters" : [ { "name" : "path",
            "value" : "../temp/lock/system"
          } ],
        "type" : "org.exoplatform.services.jcr.impl.core.lock.FileSystemLockPersister"
      },
    "timeout" : 15728640
  },
"name" : "production",
"queryHandler" : { "analyzer" : {  },
    "autoRepair" : true,
    "bufferSize" : 10,
    "cacheSize" : 1000,
    "documentOrder" : true,
```

```
      "errorLogSize" : 50,
                                                                "excerptProviderClass"          :
 "org.exoplatform.services.jcr.impl.core.query.lucene.DefaultHTMLExcerpt",
      "excludedNodeIdentifers" : null,
      "extractorBackLogSize" : 100,
      "extractorPoolSize" : 0,
      "extractorTimeout" : 100,
      "indexDir" : "../temp/jcrlucenedb/production",
                                                              "indexingConfigurationClass"          :
 "org.exoplatform.services.jcr.impl.core.query.lucene.IndexingConfigurationImpl",
      "indexingConfigurationPath" : null,
      "maxFieldLength" : 10000,
      "maxMergeDocs" : 2147483647,
      "mergeFactor" : 10,
      "minMergeDocs" : 100,
      "parameters" : [ { "name" : "index-dir",
           "value" : "../temp/jcrlucenedb/production"
         } ],
      "queryClass" : "org.exoplatform.services.jcr.impl.core.query.QueryImpl",
      "queryHandler" : null,
      "resultFetchSize" : 2147483647,
      "rootNodeIdentifer" : "00exo0jcr0root0uuid0000000000000",
      "spellCheckerClass" : null,
      "supportHighlighting" : false,
      "synonymProviderClass" : null,
      "synonymProviderConfigPath" : null,
      "type" : "org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex",
      "useCompoundFile" : false,
      "volatileIdleTime" : 3
    },
  "uniqueName" : "repository_production"
}
```


- Return when being failure

```
status code = 500          - the unknown error
failure message in response  - the description of failure
```

HTTPBackupAgent Configuration

Add the components org.exoplatform.services.jcr.ext.backup.server.HTTPBackupAgent and org.exoplatform.services.jcr.ext.backup.BackupManager to services configuration :

```
<component>
  <type>org.exoplatform.services.jcr.ext.backup.server.HTTPBackupAgent</type>
</component>

<component>
  <type>org.exoplatform.services.jcr.ext.repository.RestRepositoryService</type>
</component>

<component>
  <key>org.exoplatform.services.jcr.ext.backup.BackupManager</key>
  <type>org.exoplatform.services.jcr.ext.backup.impl.BackupManagerImpl</type>
  <init-params>
    <properties-param>
      <name>backup-properties</name>
      <property name="backup-dir" value="../temp/backup" />
    </properties-param>
  </init-params>
</component>
```

In case, if you will restore backup in same workspace (so you will drop previous workspace), you need configure RepositoryServiceConfiguration in order to save the changes of the repository configuration. For example

```
<component>
  <key>org.exoplatform.services.jcr.config.RepositoryServiceConfiguration</key>
  <type>org.exoplatform.services.jcr.impl.config.RepositoryServiceConfigurationImpl</type>
  <init-params>
    <value-param>
      <name>conf-path</name>
      <description>JCR repositories configuration file</description>
      <value>jar:/conf/portal/exo-jcr-config.xml</value>
    </value-param>
    <properties-param>
      <name>working-conf</name>
      <description>working-conf</description>
      <property name="source-name" value="jdbcjcr" />
      <property name="dialect" value="hsqldb" />
                                                <property          name="persister-class-name"
 value="org.exoplatform.services.jcr.impl.config.JDBCConfigurationPersister" />
    </properties-param>
  </init-params>
```

```
</component>
```

See the *eXo JCR Configuration article at chapter '2 Portal and Standalone configuration'* for details.

Backup Client

For GateIn should use context "/portal/rest". GateIn uses form authentication, so first you need to login (url to form authentication is http://host:port/portal/login) and then perform requests.

Backup client is support form authentication. For example call command "info" with form authentication to GateIn :

./jcrbackup.sh http://127.0.0.1:8080/portal/rest form POST "/portal/login?initialURI=/portal/private&username=root&password=gtn" info

Backup client is console application.

The backup client is http client for HTTPBackupAgent.

Command signature:

```
Help info:
 <url_basic_authentication>|<url form authentication>  <cmd>
 <url_basic_authentication>  :   http(s)//login:password@host:port/<context>

 <url form authentication>   :   http(s)//host:port/<context> "<form auth parm>"
    <form auth parm>       :   form <method> <form path>
    <method>               :   POST or GET
                        <form     path>                                    :              /path/path?<paramName1>=<paramValue1>&<paramName2>=<paramValue2>...
     Example to <url form authentication> : http://127.0.0.1:8080/portal/rest form POST "/portal/login?initialURI=/portal/private&username=root&password=gtn"

 <cmd>  :   start <repo[/ws]> <backup_dir> [<incr>]
          stop <backup_id>
          status <backup_id>
          restores <repo[/ws]>
                       restore  [remove-exists]  {{<backup_id>|<backup_set_path>}  |  {<repo[/ws]> {<backup_id>|<backup_set_path>} [<pathToConfigFile>]}}
          list [completed]
          info
          drop [force-close-session] <repo[/ws]>
          help
```

```
start          - start backup of repository or workspace
stop           - stop backup
status         - information about the current or completed backup by 'backup_id'
restores       - information about the last restore on specific repository or workspace
restore        - restore the repository or workspace from specific backup
list           - information about the current backups (in progress)
list completed - information about the completed (ready to restore) backups
info           - information about the service backup
drop           - delete the repository or workspace
help           - print help information about backup console

<repo[/ws]>         - /<reponsitory-name>[/<workspace-name>]  the repository or workspace
<backup_dir>        - path to folder for backup on remote server
<backup_id>         - the identifier for backup
<backup_set_dir>    - path to folder with backup set on remote server
<incr>              - incemental job period
<pathToConfigFile>  - path (local) to  repository or workspace configuration
remove-exists       - remove fully (db, value storage, index) exists repository/workspace
force-close-session - close opened sessions on repository or workspace.

All valid combination of parameters for command restore:
 1. restore remove-exists <repo/ws> <backup_id>       <pathToConfigFile>
 2. restore remove-exists <repo>    <backup_id>       <pathToConfigFile>
 3. restore remove-exists <repo/ws> <backup_set_path> <pathToConfigFile>
 4. restore remove-exists <repo>    <backup_set_path> <pathToConfigFile>
 5. restore remove-exists <backup_id>
 6. restore remove-exists <backup_set_path>
 7. restore <repo/ws> <backup_id>       <pathToConfigFile>
 8. restore <repo>    <backup_id>       <pathToConfigFile>
 9. restore <repo/ws> <backup_set_path> <pathToConfigFile>
10. restore <repo>    <backup_set_path> <pathToConfigFile>
11. restore <backup_id>
12. restore <backup_set_path>
```

Backup Client Usage

Building application

- Go to folder of "backup client" **${JCR-SRC-HOME}/applications/exo.jcr.applications.backupconsole** . - build the application :

```
mvn clean install -P deploy
```

- Go to **${JCR-SRC-HOME}/applications/exo.jcr.applications.backupconsole/target/backupconsole-binary** and use it.

Note

${JCR-SRC-HOME} the path where eXo JCR sources located

Running application

- Run jar

```
java -jar exo.jcr.applications.backupconsole-binary.jar <command>
```

or use jcrbackup.cmd (or .sh);

Getting information about backup service

```
jcrbackup http://root:exo@127.0.0.1:8080 info
```

Return :

```
The backup service information :
 full backup type              : org.exoplatform.services.jcr.ext.backup.impl.fs.FullBackupJob
              incremetal       backup       type                                    :
org.exoplatform.services.jcr.ext.backup.impl.fs.IncrementalBackupJob
 backup log folder             : /home/rainf0x/java/exo-working/JCR-839/new_JCR/exo-tomcat/bin/
../temp/backup
 default incremental job period : 3600
```

Starting full backup

Start full backup only on workspace "backup", the parameter <bakcup_dir> (../temp/backup) should be exists:

```
jcrbackup http://root:exo@127.0.0.1:8080 start /repository/backup ../temp/backup
```

Return :

```
Successful :
```

```
 status code = 200
```

Starting full and incremental backup on a single workspace

Start full and incremental backup on workspace "production":

```
jcrbackup http://root:exo@127.0.0.1:8080 start /repository/production ../temp/backup 10000
```

Return :

```
Successful :
 tatus code = 200
```

Getting information about the current backups (in progress)

```
jcrbackup http://root:exo@127.0.0.1:8080 list
```

Return :

```
The current backups information :
  1) Backup with id b46370107f000101014b03ea5fbe8d54 :
    repository name           : repository
    workspace name            : production
    backup type               : full + incremetal
    full backup state         : finished
    incremental backup state  : working
    started time              : Fri, 17 Apr 2009 17:03:16 EEST
  2) Backup with id b462e4427f00010101cf243b4c6015bb :
    repository name           : repository
    workspace name            : backup
    backup type               : full only
    full backup state         : finished
    started time              : Fri, 17 Apr 2009 17:02:41 EEST
```

Getting information about the current backup by 'backup_id'

```
jcrbackup http://root:exo@127.0.0.1:8080 status b46370107f000101014b03ea5fbe8d54
```

return:

```
The current backup information :
    backup id               : b46370107f000101014b03ea5fbe8d54
     backup folder             : /home/rainf0x/java/exo-working/JCR-839/new_JCR/exo-tomcat/bin/../
temp/backup
    repository name          : repository
    workspace name            : production
    backup type              : full + incremetal
    full backup state        : finished
    incremental backup state : working
    started time             : Fri, 17 Apr 2009 17:03:16 EEST
```

Stopping backup by "backup_id"

```
jcrbackup http://root:exo@127.0.0.1:8080 stop 6c302adc7f00010100df88d29535c6ee
```

Return:

```
Successful :
  status code = 200
```

Getting information about the completed (ready to restore) backups

```
jcrbackup http://root:exo@127.0.0.1:8080 list completed
```

Return:

```
The completed (ready to restore) backups information :
  1) Backup with id adf6fadc7f00010100053b2cba43513c :
```

```
   repository name          : repository
   workspace name           : backup
   backup type              : full only
   started time             : Thu, 16 Apr 2009 11:07:05 EEST

  2) Backup with id b46370107f000101014b03ea5fbe8d54 :
   repository name          : repository
   workspace name           : production
   backup type              : full + incremetal
   started time             : Fri, 17 Apr 2009 17:03:16 EEST

  3) Backup with id aec419cc7f000101004aca277b2b4e9f :
   repository name          : repository
   workspace name           : backup8
   backup type              : full only
   started time             : Thu, 16 Apr 2009 14:51:08 EEST
```

Restoring to workspace

Restore to workspace "backup3", for restore need the <backup_id> of completed backup and path to file with workspace configuration:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore /repository/backup3 6c302adc7f00010100df88d29535c6ee /home/rainf0x/java/exo-working/JCR-839/exo-jcr-config_backup3.xml
```

Return:

```
Successful :
  status code = 200
```

Getting information about the current restore

Get information about the current restore for workspace /repository/backup3:

```
jcrbackup http://root:exo@127.0.0.1:8080 restores
```

Return:

```
The current restores information :
1) Restore with id 6c302adc7f00010100df88d29535c6ee:
   full backup date       : 2009-04-03T16:34:37.394+03:00
   backup log file        : /home/rainf0x/java/exo-working/JCR-839/exo-tomcat/bin/../temp/backup/backup-6c302adc7f00010100df88d29535c6ee.xml
   repository name        : repository
   workspace name         : backup3
   backup type            : full only
   path to backup folder  : /home/rainf0x/java/exo-working/JCR-839/exo-tomcat/bin/../temp/backup
   restore state          : successful
```

Restoring workspace and remove exists workspace

Restore to workspace "backup" and remove fully (will be removed content from db, value storage, index) exists workspace, for restore need the <backup_id> of completed backup and path to file with workspace configuration:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore remove-exists /repository/backup 6c302adc7f00010100df88d29535c6ee /home/rainf0x/java/exo-working/JCR-839/exo-jcr-config_backup.xml
```

Return:

```
Successful :
  status code = 200
```

Restoring workspace from backup set

Restore to workspace "backup", for restore need the <backup_set_path> (<backup_set_path> is path to backup set folder on server side) of completed backup and path to file with workspace configuration:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore /repository/backup /tmp/123/repository_backup-20101220_114156 /home/rainf0x/java/exo-working/JCR-839/exo-jcr-config_backup.xml
```

Return:

```
Successful :
  status code = 200
```

Restoring workspace from backup set and remove exists workspace

Restore to workspace "backup" and remove fully (will be removed content from db, value storage, index) exists workspace, for restore need the <backup_set_path> (<backup_set_path> is path to backup set folder on server side) of completed backup and path to file with workspace configuration:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore remove-exists /repository/backup /repository/backup /tmp/123/repository_backup-20101220_114156 /home/rainf0x/java/exo-working/JCR-839/exo-jcr-config_backup.xml
```

Return:

```
Successful :
  status code = 200
```

Restoring workspace with original configuation

Restore to workspace "backup" with original configuration of workspace (the original configuration was stored in backup set), for restore need the <backup_id> of completed backup:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore  6c302adc7f00010100df88d29535c6ee
```

Return:

```
Successful :
  status code = 200
```

Restoring workspace with original configuation and remove exists workspace

Restore to workspace "backup" with original configuration of workspace (the original configuration was stored in backup set) and remove fully (will be removed content from db, value storage, index) exists workspace, for restore need the <backup_id> of completed backup:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore remove-exists
 6c302adc7f00010100df88d29535c6ee
```

Return:

```
Successful :
  status code = 200
```

Restoring workspace from backup set with original configuation

Restore to workspace "backup" with original configuration of workspace (the original configuration was stored in backup set), for restore need the <backup_set_path> (<backup_set_path> is path to backup set folder on server side) of completed backup:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore /tmp/123/repository_backup-
20101220_114156
```

Return:

```
Successful :
  status code = 200
```

Restoring workspace from backup set with original configuation and remove exists workspace

Restore to workspace "backup" and remove fully (will be removed content from db, value storage, index) exists workspace with original configuration of workspace (the original configuration was stored in backup set), for restore need the <backup_set_path> (<backup_set_path> is path to backup set folder on server side) of completed backup:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore remove-exists /tmp/123/repository_backup-20101220_114156
```

Return:

```
Successful :
  status code = 200
```

Restoring repository

Restore to repository "repository" , for restore need the <backup_id> of completed backup and path to file with repository configuration:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore remove-exists /repository 6c302adc7f00010100df88d29535c6ee /home/rainf0x/java/exo-working/JCR-839/exo-jcr-config.xml
```

Return:

```
Successful :
  status code = 200
```

Restoring repository and remove exists repository

Restore to repositoy "repository" and remove fully (will be removed content from db, value storage, index) exists repository, for restore need the <backup_id> of completed backup and path to file with repository configuration:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore remove-exists /repository 6c302adc7f00010100df88d29535c6ee /home/rainf0x/java/exo-working/JCR-839/exo-jcr-config.xml
```

Return:

```
Successful :
  status code = 200
```

Restoring repository from backup set

Restore to repository "repository", for restore need the <backup_set_path> (<backup_set_path> is path to backup set folder on server side) of completed backup and path to file with repository configuration:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore /repository /tmp/123/repository_repository_backup_1292833493681 /home/rainf0x/java/exo-working/JCR-839/exo-jcr-config.xml
```

Return:

```
Successful :
  status code = 200
```

Restoring repository from backup set and remove exists repository

Restore to repository "repository" and remove fully (will be removed content from db, value storage, index) exists repository, for restore need the <backup_set_path> (<backup_set_path> is path to backup set folder on server side) of completed backup and path to file with repository configuration:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore remove-exists /repository /repository/backup /tmp/123/repository_repository_backup_1292833493681 /home/rainf0x/java/exo-working/JCR-839/exo-jcr-config.xml
```

Return:

```
Successful :
  status code = 200
```

Restoring repository with original configuation

Restore to repository "repository" with original configuration of repository (the original configuration was stored in backup set), for restore need the <backup_id> of completed backup:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore  6c302adc7f00010100df88d29535c6ee
```

Return:

```
Successful :
  status code = 200
```

Restoring repository with original configuation and remove exists repository

Restore to repository "repository" with original configuration of repository (the original configuration was stored in backup set) and remove fully (will be removed content from db, value storage, index) exists repository, for restore need the <backup_id> of completed backup:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore remove-exists
 6c302adc7f00010100df88d29535c6ee
```

Return:

```
Successful :
  status code = 200
```

Restoring repository from backup set with original configuation

Restore to repository "repository" with original configuration of repository (the original configuration was stored in backup set), for restore need the <backup_set_path> (<backup_set_path> is path to backup set folder on server side) of completed backup:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore /tmp/123/
repository_repository_backup_1292833493681
```

Return:

```
Successful :
  status code = 200
```

Restoring repository from backup set with original configuation and remove exists repository

Restore to repository "repository" and remove fully (will be removed content from db, value storage, index) exists repository with original configuration of repository (the original configuration was stored in backup set), for restore need the <backup_set_path> (<backup_set_path> is path to backup set folder on server side) of completed backup:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore remove-exists /tmp/123/repository_repository_backup_1292833493681
```

Return:

```
Successful :
  status code = 200
```

Full example about creating backup and restoring it for workspace 'backup'

Creating backup

```
jcrbackup http://root:exo@127.0.0.1:8080 start /repository/backup ../temp/backup 10000
```

Return :

```
Successful :
  status code = 200
```

Getting information about current backups

```
jcrbackup http://root:exo@127.0.0.1:8080 list
```

Return :

```
The current backups information :
```

```
1) Backup with id b469ba957f0001010178febaedf20eb7 :
  repository name          : repository
  workspace name           : backup
  backup type              : full + incremetal
  full backup state        : finished
  incremental backup state  : working
  started time             : Fri, 17 Apr 2009 17:10:09 EEST
```

Stopping backup by id

Stop backup with id b469ba957f0001010178febaedf20eb7 :

```
jcrbackup http://root:exo@127.0.0.1:8080 stop b469ba957f0001010178febaedf20eb7
```

Return :

```
Successful :
  status code = 200
```

Deleting the workspace "backup" and close opened sessions on this workspace

```
jcrbackup http://root:exo@127.0.0.1:8080 drop force-close-session /repository/backup
```

Return :

```
Successful :
  status code = 200
```

Restoring the workspace "backup"

- Delete/clean the database for workspace **"backup"** : When we use "single-db", then we will run the SQL queries for clean database :

```
delete from JCR_SREF where NODE_ID in (select ID from JCR_SITEM where
 CONTAINER_NAME = 'backup')
```

```
delete from JCR_SVALUE where PROPERTY_ID in (select ID from JCR_SITEM where
 CONTAINER_NAME = 'backup')
delete from JCR_SITEM where CONTAINER_NAME='backup'
```

- Delete the value storage for workspace **"backup"**; - delete the index data for workspace **"backup"**; - restore :

```
jcrbackup http://root:exo@127.0.0.1:8080 restore /repository/backup
      b469ba957f0001010178febaedf20eb7 /home/rainf0x/java/exo-working/JCR-839/exo-jcr-
config_backup.xml
```

Return :

```
Successful :
 status code = 200
```

The /home/rainf0x/java/exo-working/JCR-839/exo-jcr-config_backup.xml content the configuration for restored workspace **"backup"**:

```
<repository-service default-repository="repository">
 <repositories>
                <repository name="repository" system-workspace="production" default-
workspace="production">
    <security-domain>exo-domain</security-domain>
    <access-control>optional</access-control>
      <authentication-policy>org.exoplatform.services.jcr.impl.core.access.JAASAuthenticator</
authentication-policy>
    <workspaces>

     <workspace name="backup">
                                                                                <container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
        <properties>
          <property name="source-name" value="jdbcjcr" />
          <property name="dialect" value="pgsql" />
          <property name="multi-db" value="false" />
          <property name="update-storage" value="false" />
          <property name="max-buffer-size" value="200k" />
          <property name="swap-directory" value="../temp/swap/backup" />
        </properties>
        <value-storages>
```

```
                                                    <value-storage       id="draft"
class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValueStorage">
            <properties>
              <property name="path" value="../temp/values/backup" />
            </properties>
            <filters>
              <filter property-type="Binary"/>
            </filters>
          </value-storage>
        </value-storages>
      </container>
      <initializer class="org.exoplatform.services.jcr.impl.core.ScratchWorkspaceInitializer">
        <properties>
          <property name="root-nodetype" value="nt:unstructured" />
        </properties>
      </initializer>
                                                    <cache          enabled="true"
class="org.exoplatform.services.jcr.impl.dataflow.persistent.LinkedWorkspaceStorageCacheImpl">
        <properties>
          <property name="max-size" value="10k" />
          <property name="live-time" value="1h" />
        </properties>
      </cache>
      <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
        <properties>
          <property name="index-dir" value="../temp/jcrlucenedb/backup" />
        </properties>
      </query-handler>
    </workspace>
  </workspaces>
 </repository>
</repositories>
</repository-service>
```

Getting information about restore for workspace /repository/backup

```
jcrbackup http://root:exo@127.0.0.1:8080 restores /repository/backup
```

Return:

```
The current restores information :
  Restore with id b469ba957f0001010178febaedf20eb7:
     backup folder            : /home/rainf0x/java/exo-working/JCR-839/new_JCR/exo-tomcat/bin/../
temp/backup
    repository name         : repository
    workspace name          : backup
    backup type             : full + incremetal
    restore state           : successful
    started time            : Fri, 17 Apr 2009 16:38:00 EEST
    finished time           : Fri, 17 Apr 2009 16:38:00 EEST
```

Full example about creating backup and restoring it for repository 'repository'

Note

If delete default repository that should be restored repository with name as default repository.

This usecase needs RestRepositoryService enabled. (Deleting the repository needs it)

```
<component>
   <type>org.exoplatform.services.jcr.ext.repository.RestRepositoryService</type>
</component>
```

Creating backup

```
jcrbackup http://root:exo@127.0.0.1:8080 start /repository ../temp/backup 10000
```

Return :

```
Successful :
  status code = 200
```

Getting information about current backups

```
jcrbackup http://root:exo@127.0.0.1:8080 list
```

Return :

```
The current backups information :
  1) Repository backup with id 9a4d40fb7f0000012ec8f0a4ec70b3da :
    repository name            : repository
    backup type                : full + incremetal
    full backups state         : finished
    incremental backups state  : working
    started time               : Mon, 11 Oct 2010 10:59:35 EEST
```

Stopping backup by id

Stop backup with id 9a4d40fb7f0000012ec8f0a4ec70b3da :

```
jcrbackup http://root:exo@127.0.0.1:8080 stop 9a4d40fb7f0000012ec8f0a4ec70b3da
```

Return :

```
Successful :
  status code = 200
```

Deleting the repository "repository" and close all opened sessions

```
jcrbackup http://root:exo@127.0.0.1:8080 drop force-close-session /repository
```

Return :

```
Successful :
  status code = 200
```

Restoring the repository "repository"

- Delete/clean the database for workspace **"repository"**: When we use "single-db", then we will run the SQL queries for clean database :

```
drop table JCR_SREF;
drop table JCR_SVALUE;
drop table JCR_SITEM;
```

- Delete the value storage for repository **"repository"**;
- Delete the index data for repository **"repository"**;
- Restore:

```
jcrbackup http://root:exo@127.0.0.1:8080 restore /repository 9a6dba327f000001325dfb228a181b07 /home/rainf0x/exo-jcr-config_backup.xml
```

Return :

```
Successful :
 status code = 200
```

The /home/rainf0x/exo-jcr-config_backup.xml content the configuration for restored repository **"repository"**:

```
<repository-service default-repository="repository">
  <repositories>
                    <repository name="repository" system-workspace="production" default-workspace="production">
      <security-domain>exo-domain</security-domain>
      <access-control>optional</access-control>
       <authentication-policy>org.exoplatform.services.jcr.impl.core.access.JAASAuthenticator</authentication-policy>
      <workspaces>
        <workspace name="production">
          <!-- for system storage -->
                                                                                <container class="org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.CQJDBCWorkspaceDataContainer">
            <properties>
              <property name="source-name" value="jdbcjcr" />
```

```
            <property name="multi-db" value="false" />
            <property name="update-storage" value="false" />
            <property name="max-buffer-size" value="200k" />
            <property name="swap-directory" value="../temp/swap/production" />
          </properties>
          <value-storages>
                                                                <value-storage     id="system"
class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValueStorage">
              <properties>
                <property name="path" value="../temp/values/production" />
              </properties>
              <filters>
                <filter property-type="Binary" />
              </filters>
            </value-storage>
          </value-storages>
        </container>
        <initializer class="org.exoplatform.services.jcr.impl.core.ScratchWorkspaceInitializer">
          <properties>
            <property name="root-nodetype" value="nt:unstructured" />
          </properties>
        </initializer>

                                                                    <cache
class="org.exoplatform.services.jcr.impl.dataflow.persistent.LinkedWorkspaceStorageCacheImpl">
          <properties>
            <property name="max-size" value="10k" />
            <property name="live-time" value="1h" />
          </properties>
        </cache>
      <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
          <properties>
            <property name="index-dir" value="../temp/jcrlucenedb/production" />
          </properties>
        </query-handler>
        <lock-manager>
          <time-out>15m</time-out>
        <persister class="org.exoplatform.services.jcr.impl.core.lock.FileSystemLockPersister">
          <properties>
            <property name="path" value="../temp/lock/system" />
          </properties>
        </persister>
      </lock-manager>
    </workspace>
```

```
        <workspace name="backup">
                                                                                <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.CQJDBCWorkspaceDataContainer">
              <properties>
                <property name="source-name" value="jdbcjcr" />
                <property name="multi-db" value="false" />
                <property name="update-storage" value="false" />
                <property name="max-buffer-size" value="200k" />
                <property name="swap-directory" value="../temp/swap/backup" />
              </properties>
              <value-storages>
                                                                    <value-storage     id="draft"
class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValueStorage">
                  <properties>
                    <property name="path" value="../temp/values/backup" />
                  </properties>
                  <filters>
                    <filter property-type="Binary" />
                  </filters>
                </value-storage>
              </value-storages>
            </container>
            <initializer class="org.exoplatform.services.jcr.impl.core.ScratchWorkspaceInitializer">
              <properties>
                <property name="root-nodetype" value="nt:unstructured" />
              </properties>
            </initializer>

                                                                        <cache
class="org.exoplatform.services.jcr.impl.dataflow.persistent.LinkedWorkspaceStorageCacheImpl">
              <properties>
                <property name="max-size" value="10k" />
                <property name="live-time" value="1h" />
              </properties>
            </cache>
          <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
              <properties>
                <property name="index-dir" value="../temp/jcrlucenedb/backup" />
              </properties>
            </query-handler>
          </workspace>

          <workspace name="digital-assets">
```

```
                                                                                <container
class="org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.CQJDBCWorkspaceDataContainer">
            <properties>
              <property name="source-name" value="jdbcjcr" />
              <property name="multi-db" value="false" />
              <property name="update-storage" value="false" />
              <property name="max-buffer-size" value="200k" />
              <property name="swap-directory" value="../temp/swap/digital-assets" />
            </properties>
            <value-storages>
                                                              <value-storage    id="digital-assets"
class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValueStorage">
                <properties>
                  <property name="path" value="../temp/values/digital-assets" />
                </properties>
                <filters>
                  <filter property-type="Binary" />
                </filters>
              </value-storage>
            </value-storages>
          </container>
          <initializer class="org.exoplatform.services.jcr.impl.core.ScratchWorkspaceInitializer">
            <properties>
              <property name="root-nodetype" value="nt:folder" />
            </properties>
          </initializer>

                                                                    <cache
class="org.exoplatform.services.jcr.impl.dataflow.persistent.LinkedWorkspaceStorageCacheImpl">
            <properties>
              <property name="max-size" value="5k" />
              <property name="live-time" value="15m" />
            </properties>
          </cache>
        <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
            <properties>
              <property name="index-dir" value="../temp/jcrlucenedb/digital-assets" />
            </properties>
          </query-handler>
        </workspace>
      </workspaces>
    </repository>
  </repositories>
```

```
</repository-service>
```

Getting information about restore for repository 'repository'

```
jcrbackup http://root:exo@127.0.0.1:8080 restores /repository
```

Return:

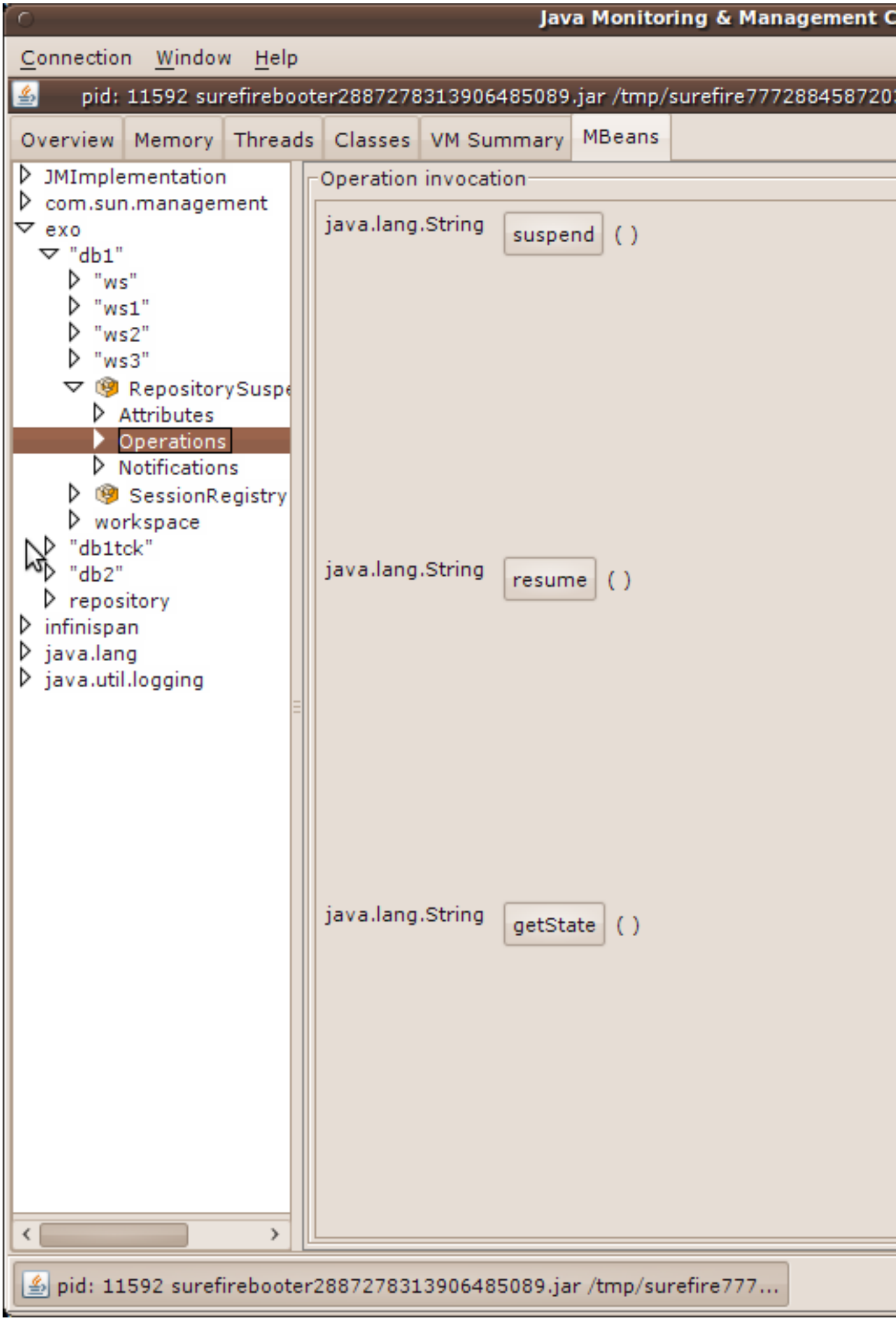
```
  Repository restore with id 9a6dba327f000001325dfb228a181b07:
    backup folder         : /home/rainf0x/java/exo-working/JCR-1459/exo-tomcat/bin/../temp/backup/
repository_repository_backup_1286786103858
     repository name         : repository
     backup type             : full + incremetal
     restore state           : successful
     started time            : Mon, 11 Oct 2010 11:51:15 EEST
     finished time           : Mon, 11 Oct 2010 11:51:17 EEST
```

Use external backup tool

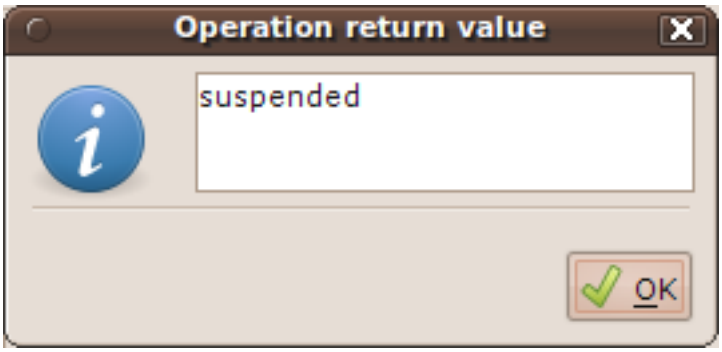
Repository suspending

To have the repository content consistent with the search index and value storate, the repository should be suspened. It means all working threads are suspended until resume operation performed. Index will be flushed.

JCR provides ability to suspend repository via JMX.


Java Monitoring & Management C
Connection
Window
Help
pid: 11592 surefirebooter2887278313906485089.jar /tmp/surefire777288458720
Overview
Memory
Threads
Classes
VM Summary
MBeans
JMImplementation
com.sun.management
exo
"db1"
"ws"
"ws1"
"ws2"
"ws3"
RepositorySuspe
Attributes
Operations
Notifications
SessionRegistry
workspace
"db1tck"
"db2"
repository
infinispan
java.lang
java.util.logging
Operation invocation
java.lang.String
suspend
( )
java.lang.String
resume
( )
java.lang.String
getState
( )
pid: 11592 surefirebooter2887278313906485089.jar /tmp/surefire777...

To suspend repository just need to invoke suspend() operation. The returned result will be "suspended" if everything passed successfully.



The result "undefined" means not all components successfully suspended, check console to see stacktraces.

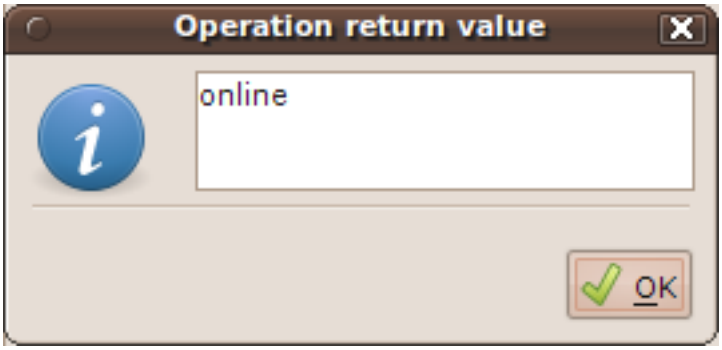
Backup

Now we can backup content manually or using third part software. We need to backup:

- Database
- Lucene index
- Value storage (if configured)

Repository resuming

Once backup is done need to invoke resume() operation to switch repository back to online. The returned result will be "online".



eXo JCR statistics

Statistics on the Database Access Layer

In order to have a better idea of the time spent into the database access layer, it can be interesting to get some statistics on that part of the code, knowing that most of the time spent into eXo JCR is mainly the database access. This statistics will then allow you to identify without using any profiler what is normally slow in this layer, which could help to fix the problem quickly.

In case you use `org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.CQJDBCWorkspaceDataContainer` or `org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer` as `WorkspaceDataContainer`, you can get statistics on the time spent into the database access layer. The database access layer (in eXo JCR) is represented by the methods of the interface `org.exoplatform.services.jcr.storage.WorkspaceStorageConnection`, so for all the methods defined in this interface, we can have the following figures:

- The minimum time spent into the method.
- The maximum time spent into the method.
- The average time spent into the method.
- The total amount of time spent into the method.
- The total amount of time the method has been called.

Those figures are also available globaly for all the methods which gives us the global behavior of this layer.

If you want to enable the statistics, you just need to set the JVM parameter called *JDBCWorkspaceDataContainer.statistics.enabled* to *true*. The corresponding CSV file is *StatisticsJDBCStorageConnection-${creation-timestamp}.csv* for more details about how the csv files are managed, please refer to the section dedicated to the statistics manager.

The format of each column header is ${method-alias}-${metric-alias}. The metric alias are described in the statistics manager section.

The name of the category of statistics corresponding to these statistics is *JDBCStorageConnection*, this name is mostly needed to access to the statistics through JMX.

Table 37.1. Method Alias

global	This is the alias for all the methods.

getItemDataById	This is the alias for the method *getItemData(String identifier).*
getItemDataByNodeDataNQPathEntry	This is the alias for the method *getItemData(NodeData parentData, QPathEntry name).*
getChildNodesData	This is the alias for the method *getChildNodesData(NodeData parent).*
getChildNodesCount	This is the alias for the method *getChildNodesCount(NodeData parent).*
getChildPropertiesData	This is the alias for the method *getChildPropertiesData(NodeData parent).*
listChildPropertiesData	This is the alias for the method *listChildPropertiesData(NodeData parent).*
getReferencesData	This is the alias for the method *getReferencesData(String nodeIdentifier).*
commit	This is the alias for the method *commit().*
addNodeData	This is the alias for the method *add(NodeData data).*
addPropertyData	This is the alias for the method *add(PropertyData data).*
updateNodeData	This is the alias for the method *update(NodeData data).*
updatePropertyData	This is the alias for the method *update(PropertyData data).*
deleteNodeData	This is the alias for the method *delete(NodeData data).*
deletePropertyData	This is the alias for the method *delete(PropertyData data).*
renameNodeData	This is the alias for the method *rename(NodeData data).*
rollback	This is the alias for the method *rollback().*
isOpened	This is the alias for the method *isOpened().*
close	This is the alias for the method *close().*

Statistics on the JCR API accesses

In order to know exactly how your application uses eXo JCR, it can be interesting to register all the JCR API accesses in order to easily create real life test scenario based on pure JCR calls and also to tune your eXo JCR to better fit your requirements.

In order to allow you to specify the configuration which part of eXo JCR needs to be monitored whithout applying any changes in your code and/or building anything, we choose to rely on the Load-time Weaving proposed by AspectJ.

To enable this feature, you will have to add in your classpath the following jar files:

- *exo.jcr.component.statistics-X.Y.Z*.jar corresponding to your eXo JCR version that you can get from the jboss maven repository *http://repository.jboss.com/maven2/org/exoplatform/jcr/exo.jcr.component.statistics*.
- aspectjrt-1.6.8.jar that you can get from the main maven repository `http://repo2.maven.org/maven2/org/aspectj/aspectjrt`.

You will also need to get aspectjweaver-1.6.8.jar from the main maven repository *http://repo2.maven.org/maven2/org/aspectj/aspectjweaver*. At this stage, to enable the statistics on the JCR API accesses, you will need to add the JVM parameter *-javaagent:${pathto}/aspectjweaver-1.6.8.jar* to your command line, for more details please refer to *http://www.eclipse.org/aspectj/doc/released/devguide/ltw-configuration.html*.

By default, the configuration will collect statistcs on all the methods of the internal interfaces *org.exoplatform.services.jcr.core.ExtendedSession* and *org.exoplatform.services.jcr.core.ExtendedNode*, and the JCR API interface *javax.jcr.Property*. To add and/or remove some interfaces to monitor, you have two configuration files to change that are bundled into the jar *exo.jcr.component.statistics-X.Y.Z*.jar, which are *conf/configuration.xml* and *META-INF/aop.xml*.

The file content below is the content of *conf/configuration.xml* that you will need to modify to add and/or remove the full qualified name of the interfaces to monitor, into the list of parameter values of the init param called *targetInterfaces*.

```
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
 xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">

 <component>
  <type>org.exoplatform.services.jcr.statistics.JCRAPIAspectConfig</type>
  <init-params>
   <values-param>
    <name>targetInterfaces</name>
    <value>org.exoplatform.services.jcr.core.ExtendedSession</value>
    <value>org.exoplatform.services.jcr.core.ExtendedNode</value>
    <value>javax.jcr.Property</value>
   </values-param>
  </init-params>
```

```
  </component>
</configuration>
```

The file content below is the content of *META-INF/aop.xml* that you will to need to modify to add and/or remove the full qualified name of the interfaces to monitor, into the expression filter of the pointcut called *JCRAPIPointcut*. As you can see below, by default only JCR API calls from the exoplatform packages are took into account, don't hesistate to modify this filter to add your own package names.

```
<aspectj>
  <aspects>
            <concrete-aspect    name="org.exoplatform.services.jcr.statistics.JCRAPIAspectImpl"
 extends="org.exoplatform.services.jcr.statistics.JCRAPIAspect">
     <pointcut name="JCRAPIPointcut"
                              expression="(target(org.exoplatform.services.jcr.core.ExtendedSession)
     ||     target(org.exoplatform.services.jcr.core.ExtendedNode)     ||     target(javax.jcr.Property))
 &amp;amp;&amp;amp; call(public * *(..))" />
    </concrete-aspect>
  </aspects>
  <weaver options="-XnoInline">
    <include within="org.exoplatform..*" />
  </weaver>
</aspectj>
```

The corresponding CSV files are of type *Statistics${interface-name}-${creation-timestamp}.csv* for more details about how the csv files are managed, please refer to the section dedicated to the statistics manager.

The format of each column header is ${method-alias}-${metric-alias}. The method alias will be of type ${method-name}(list of parameter types separeted by ; to be compatible with the CSV format).

The metric alias are described in the statistics manager section.

The name of the category of statistics corresponding to these statistics is the simple name of the monitored interface (e.g. ExtendedSession for org.exoplatform.services.jcr.core.ExtendedSession), this name is mostly needed to access to the statistics through JMX.
Please note that this feature will affect the performances of eXo JCR so it must be used with caution.

Statistics Manager

The statistics manager manages all the statistics provided by eXo JCR, it is responsible of printing the data into the CSV files and also exposing the statistics through JMX and/or Rest.

The statistics manager will create all the CSV files for each category of statistics that it manages, the format of those files is *Statistics${category-name}-${creation-timestamp}.csv*. Those files will be created into the user directory if it is possible otherwise it will create them into the temporary directory. The format of those files is CSV (i.e. Comma-Seperated Values), one new line will be added regularily (every 5 seconds by default) and one last line will be added at JVM exit. Each line, will be composed of the 5 figures described below for each method and globaly for all the methods.

Table 37.2. Metric Alias

Min	The minimum time spent into the method expressed in milliseconds.
Max	The maximum time spent into the method expressed in milliseconds.
Total	The total amount of time spent into the method expressed in milliseconds.
Avg	The average time spent into the method expressed in milliseconds.
Times	The total amount of times the method has been called.

You can disable the persistence of the statistics by setting the JVM parameter called *JCRStatisticsManager.persistence.enabled* to *false*, by default, it is set to *true*. You can aslo define the period of time between each record (i.e. line of data into the file) by setting the JVM parameter called *JCRStatisticsManager.persistence.timeout* to your expected value expressed in milliseconds, by default it is set to *5000*.

You can also access to the statistics thanks to JMX, the available methods are the following:

Table 37.3. JMX Methods

getMin	Give the minimum time spent into the method corresponding to the given category name and statistics name. The expected arguments are the name of the category of statistics (e.g. JDBCStorageConnection) and the name of the expected method or global for the global value.
getMax	Give the maximum time spent into the method corresponding to the given category name and statistics name. The expected arguments are the name of the category of statistics (e.g. JDBCStorageConnection) and the name of the expected method or global for the global value.
getTotal	Give the total amount of time spent into the method corresponding to the given category

	name and statistics name. The expected arguments are the name of the category of statistics (e.g. JDBCStorageConnection) and the name of the expected method or global for the global value.
getAvg	Give the average time spent into the method corresponding to the given category name and statistics name. The expected arguments are the name of the category of statistics (e.g. JDBCStorageConnection) and the name of the expected method or global for the global value.
getTimes	Give the total amount of times the method has been called corresponding to the given ,category name and statistics name. The expected arguments are the name of the category of statistics (e.g. JDBCStorageConnection) and the name of the expected method or global for the global value.
reset	Reset the statistics for the given category name and statistics name. The expected arguments are the name of the category of statistics (e.g. JDBCStorageConnection) and the name of the expected method or global for the global value.
resetAll	Reset all the statistics for the given category name. The expected argument is the name of the category of statistics (e.g. JDBCStorageConnection).

The full name of the related MBean is *exo:service=statistic, view=jcr*.

JTA

eXo JCR supports the Java Transaction API out of the box. If a *TransactionService* has been defined (refer to the chapter about the TransactionService for more details) at session save, it checks if a global transaction is active and if so, it automatically enrolles the JCR session in the global transaction. If you intend to use a managed data source, you will have to configure the service *DataSourceProvider* (for more details please refer to the corresponding chapter).

The JCA Resource Adapter

Overview

eXo JCR supports *J2EE Connector Architecture* 1.5, thus If you would like to delegate the JCR Session lifecycle to your application server, you can use the JCA Resource Adapter for eXo JCR if your application server supports JCA 1.5. This adapter only supports XA Transaction, in other words you cannot use it for local transactions. Since the JCR Sessions have not been designed to be shareable, the session pooling is simply not covered by the adapter.

The *SessionFactory*

The equivalent of the *javax.resource.cci.ConnectionFactory* in JCA terminology is *org.exoplatform.connectors.jcr.adapter.SessionFactory* in the context of eXo JCR, the resource that you will get thanks to a JNDI lookup is of type *SessionFactory* and provides the following methods:

```
/**
 * Get a JCR session corresponding to the repository
 * defined in the configuration and the default workspace.
 * @return a JCR session corresponding to the criteria
 * @throws RepositoryException if the session could not be created
 */
Session getSession() throws RepositoryException;

/**
 * Get a JCR session corresponding to the repository
 * defined in the configuration and the default workspace, using
 * the given user name and password.
 * @param userName the user name to use for the authentication
 * @param password the password to use for the authentication
 * @return a JCR session corresponding to the criteria
 * @throws RepositoryException if the session could not be created
 */
Session getSession(String userName, String password) throws RepositoryException;

/**
 * Get a JCR session corresponding to the repository
 * defined in the configuration and the given workspace.
 * @param workspace the name of the expected workspace
 * @return a JCR session corresponding to the criteria
 * @throws RepositoryException if the session could not be created
 */
```

```
  Session getSession(String workspace) throws RepositoryException;

  /**
   * Get a JCR session corresponding to the repository
   * defined in the configuration and the given workspace, using
   * the given user name and password.
   * @param workspace the name of the expected workspace
   * @param userName the user name to use for the authentication
   * @param password the password to use for the authentication
   * @return a JCR session corresponding to the criteria
   * @throws RepositoryException if the session could not be created
   */
      Session getSession(String workspace, String userName, String password) throws
 RepositoryException;
```

Configuration

Table 39.1. Configuration Properties

PortalContainer	In case of the portal mode, if no portal container can be found in the context of the request, the adapter will use the value of this parameter to get the name of the expected portal container to create the JCR sessions. In case of a standalone mode, this parameter is not used. This parameter is optional, by default the default portal container will be used.
Repository	The repository name used to create JCR sessions. This parameter is optional, by default the current repository will be used.

Deployment

In case of the standalone mode where the JCR and its dependencies are not provided, you will need to deploy the whole ear file corresponding to the artifactId *exo.jcr.ear* and groupId *org.exoplatform.jcr*, the rar file is embedded into the ear file. In case the JCR and its dependencies are provided like when you use it with gateIn for example, you will need to deploy only the rar file corresponding to the artifactId *exo.jcr.connectors.jca* and groupId *org.exoplatform.jcr*.

Then you will need to configure the connector itself, for example for JBoss AS, you need to create in your deploy directory a file of type **-ds.xml* (jcr-ds.xml for example) with the following content:

```
<connection-factories>
```

```
  <tx-connection-factory>
    <jndi-name>jcr/repository</jndi-name>
    <xa-transaction/>
    <!-- The rar name will be exo.jcr.connectors.jca.X.Y.Z.rar in case you deploy only the rar file -->
    <rar-name>exo.jcr.ear.ear#exo-jcr.rar</rar-name>
    <adapter-display-name>eXo JCR Adapter</adapter-display-name>
      <connection-definition>org.exoplatform.connectors.jcr.adapter.SessionFactory</connection-
definition>
    <!--
    <config-property name="PortalContainer" type="java.lang.String">portal</config-property>
    -->
    <config-property name="Repository" type="java.lang.String">repository</config-property>
  </tx-connection-factory>
</connection-factories>
```

Access Control

eXo JCR is a complete implementation of the standard JSR 170: *Content Repository for Java TM Technology API* [http://jcp.org/en/jsr/detail?id=170], including **Level 1, Level 2 and Additional Features** specified in the JCR Specification.

Standard Action Permissions

The JCR specification (JSR 170) does not have many requirements about Access Control. It only requires the implementation of the Session.checkPermission(String absPath, String actions) method. This method checks if a current session has permissions to perform some actions on absPath:

- absPath : The string representation of a JCR absolute path.
- actions : eXo JCR interprets this string as a comma separated the list of individual action names, such as the 4 types defined in JSR 170 :
  - **add_node** : Permission to add a node.
  - **set_property** : Permission to set a property.
  - **remove** : Permission to remove an item (node or property).
  - **read** : Permission to retrieve a node or read a property value.

For example :

- session.checkPermission("/Groups/organization", "add_node,set_property") will check if the session is allowed to add a child node to "organization" and to modify its properties. If one of the two permissions is denied, an AccessDeniedException is thrown.
- session.checkPermission("/Groups/organization/exo:name", "read,set_property") will check if the session is allowed to read and change the "exo:name" property of the "organization" node.
- session.checkPermission("/Groups/organization/exo:name", "remove") will check if the session allowed to remove "exo:name" property or node.

eXo Access Control

The JSR170 specification does not define how permissions are managed or checked. So eXo JCR has implemented its own proprietary extension to manage and check permissions on nodes. In essence, this extension uses an *Access Control List (ACL)* [http://en.wikipedia.org/wiki/Access_control_list] policy model applied to eXo Organization model (see eXo Platform Organization Service).

Principal and Identity

At the heart of eXo Access Control, is the notion of the **identity** concept. Access to JCR is made through sessions acquired against a repository. Sessions can be authenticated through the standard (but optional) repository login mechanism. Each session is associated with a **principal**. The principal is an authenticated user or group that may act on JCR data. The identity is a string identifying this **group or user**.'

There are 3 reserved identities that have special meanings in eXo JCR:

- **any** : represents any authenticated session.
- **anonim** : represents a principal for non authenticated sessions. (No error, it's really "__anonim".)
- **system** : represents a principal for system sessions, typically used for administrative purposes. System session has full access (all permissions) to all nodes; therefore be careful when working with system sessions.

ACL

An access control list (ACL) is a list of permissions attached to an object. An ACL specifies which users, groups or system processes are granted access to JCR nodes, as well as what operations are allowed to be performed on given objects.

eXo JCR Access Control is based on two facets applied to nodes :

- **Privilegeable** : Means that the user or group (also called principal) needs the appropriate privileges to access to this node. The privileges are defined as (positive) permissions that are granted to users or groups.
- **Ownable** : The node has an **owner**. The owner has always **full access** (all permissions) to the node, independent of the privilegeable facet.

Privilegeable

A privilegeable node defines the permissions required for actions on this node. For this purpose, it contains an ACL.

At JCR level, this is implemented by an **exo:privilegeable** mixin.

```
<nodeType name="exo:privilegeable" isMixin="true" hasOrderableChildNodes="false"
 primaryItemName="">
  <propertyDefinitions>
    <propertyDefinition name="exo:permissions" requiredType="Permission" autoCreated="true"
 mandatory="true"
```

```
                onParentVersion="COPY" protected="true" multiple="true">
      <valueConstraints/>
    </propertyDefinition>
  </propertyDefinitions>
</nodeType>
```

A privilegeable node can have multiple exo:permissions values. The type of these values is the eXo JCR specific Permission type. The Permission type contains a list of ACL.

The possible values are corresponding to JCR standard actions:

- **read**: The node or its properties can be read.
- **remove**: The node or its properties can be removed.
- **add_node** : Child nodes can be added to this node.
- **set_property** : The node's properties can be modified, added or removed.

Ownable

An ownable node defines an owner identity. The **owner** has always **full privileges**. These privileges are independent of the permissions set by exo:permissions. At JCR level, the ownership is implemented by an **exo:owneable** mixin. This mixin holds an owner property.

```
<nodeType name="exo:owneable" isMixin="true" hasOrderableChildNodes="false"
 primaryItemName="">
  <propertyDefinitions>
        <propertyDefinition name="exo:owner" requiredType="String" autoCreated="true"
 mandatory="true" onParentVersion="COPY"
                protected="true" multiple="false">
      <valueConstraints/>
    </propertyDefinition>
  </propertyDefinitions>
</nodeType>
```

The exo:owner property value contains exactly one identity string value. There might be a long list of different permissions for different identities (user or groups). All permissions are always positive permissions; denials are not possible. When checking a permission of an action, it's therefore perfectly sufficient that the principal of a session belongs to the groups to which the concerned action is granted.

ACL Inheritance

To grant or deny access to a node, eXo JCR applies a privilege resolving logic at node access time.

If a node is **privilegeable**, the node's ACL is used exclusively. If the ACL does not match the principal's identity, the principal has no access (except the owner of the node).

Non-privilegeable nodes inherit permissions from their parent node. If the parent node is not privilegeable either, the resolving logic looks further up the node hierarchy and stops with the first privilegeable ancestor of the current node. All nodes potentially inherit from the **workspace** root node.

The owner of a node is inherited in accordance with the same logic: If the node has no owner, the owner information of the closest owneable ancestor is inherited.

This inheritance is implemented by browsing up the node's hierarchy. At access time, if the node does not have owner or permissions, the system looks up into the node's ancestor hierarchy for the **first** ACL.

Default ACL of the root node

When no matching ACL is found in the ancestor hierarchy, the system may end up looking at the root node's ACL. As ACL are optional, even for the root node, if the root node has no ACL, the following rule is ultimately applied to resolve privileges:

- **any** identity (any authenticated session) is granted all permissions

Notes

Access Control nodetypes are not extendible: The access control mechanism works for **exo:owneable** and **exo:privilegeable** nodetypes only, not for their subtypes! So you cannot extend those nodetypes.

Autocreation: By default, newly created nodes are neither **exo:privilegeable** nor **exo:owneable** but it is possible to configure the repository to auto-create **exo:privilegeable** or/and **exo:owneable** thanks to eXo's JCR interceptors extension (see *JCR Extensions*)

OR-based Privilege Inheritance: Note, that eXo's Access Control implementation supports a privilege inheritance that follows a strategy of either...or/ and has only an ALLOW privilege mechanism (there is no DENY feature). This means that a session is allowed to perform some operations on some nodes if its identity has an appropriate permission assigned to this node. Only if there is no exo:permission property assigned to the node itself, the permissions of the node's ancestors are used.

Example

XML Example

In the following example, you see a node named "Politics" which contains two nodes named "Cats" and "Dogs".

Note

These examples are exported from eXo DMS using the \"document view\" representation of JCR. Each value of a multi-value property is separated by a whitespace, each whitespace is escaped by *x0020*.

```
<Politics jcr:primaryType="nt:unstructured" jcr:mixinTypes="exo:owneable exo:datetime
 exo:privilegeable" exo:dateCreated="2009-10-08T18:02:43.687+02:00"
exo:dateModified="2009-10-08T18:02:43.703+02:00"
exo:owner="root"
exo:permissions="any_x0020_read *:/platform/administrators_x0020_read *:/platform/
administrators_x0020_add_node *:/platform/administrators_x0020_set_property *:/platform/
administrators_x0020_remove">

<Cats jcr:primaryType="exo:article"
jcr:mixinTypes="exo:owneable"
exo:owner="marry"
exo:summary="The_x0020_secret_x0020_power_x0020_of_x0020_cats_x0020_influences_x0020_the_x0020_lea

exo:text="" exo:title="Cats_x0020_rule_x0020_the_x0020_world" />

<Dogs jcr:primaryType="exo:article"
jcr:mixinTypes="exo:privilegeable"
exo:permissions="manager:/organization_x0020_read manager:/
organization_x0020_set_property"
exo:summary="Dogs"
exo:text="" exo:title="Dogs_x0020_are_x0020_friends" />

</Politics>
```

The "Politics" node is **exo:owneable** and **exo:privilegeable**. It has both an **exo:owner** property and an **exo:permissions** property. There is an **exo:owner="root"** property so that the user root is the owner. In the exo:permissions value, you can see the ACL that is a list of access controls. In this example, the group ***:/platform/administrators** has all rights on this node (remember that the "*****" means any kind of membership). **any** means that any users also have the read permission.s

As you see in the jcr:mixinTypes property, the "Cats" node is **exo:owneable** and there is an **exo:owner="marry"** property so that the user marry is the owner. The "Cats" node is **not exo:privilegeable** and has **no exo:permissions**. In this case, we can see the **inheritance mechanism** here is that the "Cats" node has the same permissions as "Politics" node.

Finally, the "Dogs" node is also a child node of "Politics". This node is **not exo:owneable** and inherits the owner of the "Politics" node (which is the user root). Otherwise, "Dogs" is **exo:privilegeable** and therefore, it has its own **exo:permissions**. That means only the users

having a "manager" role in the group "/organization" and the user "root" have the rights to access this node.

Inheritance Examples

Here is an example showing the accessibility of two nodes (to show inheritance) for two sample users named **manager** and **user**:

The "+" symbol means that there is a child node "exo:owneable".

Default owner - system

Default permissions - any:read,set_property,remove

"." - means inherited from a root node /

Node	owner	exo:owneable	exo:privilegeable	us	
				read	set_property
/node	system	-	-	yes	yes
/node/subNode	system	-	-	yes	yes
/node	manager	+	-	yes	yes
/node/subNode	system	inherited from /node	-	yes	yes
/node	system	-	any:read, remove	yes	no
/node/subNode	system	-	inherited from /node	yes	no
/node	manager	+	any:read, remove	yes	no
/node/subNode	system	inherited from /node	inherited from /node	yes	no

Permission validation

This session describes how permission is validated for different JCR actions.

- **read node**: Check the read permission on a target node.

For example: Read /node1/**subnode** node, JCR will check the "read" permission exactly on "subnode".

- **read property** : Check the read permission on a parent node.

  For example: Read /**node1**/myprop - JCR will check the "read" permission on "node1".

- **add node**: Check add_node on a parent node.

  For example: Add /**node1**/subnode node, JCR will check the "add_node" permission on "node1".

- **set property**: set_property on a parent node.

  For example: Try to set /**node1**/myprop property, JCR will check the "set_property" permission on "node1".

- **remove node**: Check the remove permission on a target node.

  For example: Try to remove /node1/**subnode** node, JCR will check the "remove" permission on "subnode".

- **remove property**: Check the remove permission on a parent node.

  For example: Try to remove /**node1**/myprop property, JCR will check the "remove" permission on "node1".

- **add mixin**: Check the "add_node" and "set_property" permission on a target node.

  For example: Try to add mixin to /node1/**subnode** node, JCR will check the "add_node" and "set_property" permission on "subnode".

Note

The behavior of the permission "remove" and "add mixin" validation has changed since JCR 1.12.6-GA. The old behavior is:

- **remove node**: Check the remove permission on a parent node.

  For example: Try to remove /**node1**/subnode node, JCR will check the "remove" permission on "node1".

- **add mixin**: Check the "add_node" and "set_property" permission on a parent node.

  For example: Try to add mixin to /**node1**/subnode node, JCR will check the "add_node" and "set_property" permission on "node1".

Java API

eXo JCR's ExtendedNode interface which extends javax.jcr.Node interface provides additional methods for Access Control management.

Table 40.1. Additional methods

Method signature	Description
void setPermissions(Map<String, String[]> permissions)	Assigns a set of Permissions to a node
void setPermission(String identity, String[] permission)	Assigns some Identities' Permission to a node
void removePermission(String identity)	Removes an Identity's Permission
void removePermission(String identity, String permission)	Removes the specified permission for a particular identity
void clearACL()	Clears the current ACL so it becomes default
AccessControlList getACL()	Returns the current ACL
void checkPermission(String actions)	Checks Permission (AccessDeniedException will be thrown if denied)

The "identity" parameter is a user or a group name. The permissions are the literal strings of the standard action permissions (add_node, set_property, remove, read).

Access Control Extension

Prerequisites

This is an extension of eXo JCR Access Control features. Please read *Access Control* and *JCR Extensions* topics first.

Overview

An extended Access Control system consists of:

- Specifically configured custom **ExtendedAccessManager** which is called by eXo JCR internals to check if user's Session (user) has some privilege to perform some operation or not.
- The **Action** sets a thread local **InvocationContext** at runtime, the InvocationContext instance is then used by the ExtendedAccessManager in handling permissions of the current Session.
- **InvocationContext** is a collection of properties which reflect the state of a current Session. At present, it contains: the type of the current operation on Session (event), current Item (javax.jcr.Item) on which this operation is performed and the current eXo Container

Access Context Action

SetAccessContextAction implements Action and may be called by SessionActionInterceptor as a reaction of some events - usually before writing methods and after reading (getNode(), getProperty() etc). This SetAccessContextAction calls the AccessManager.setContext(InvocationContext context) method which sets the ThreadLocal invocation context for the current call.

Action's Configuration may look like as the following:

```
<value>
  <object type="org.exoplatform.services.jcr.impl.ext.action.ActionConfiguration">
    <field  name="eventTypes"><string>addNode,read</string></field>
    <field  name="workspace"><string>production</string></field >
                                                                    <field

string></field>
  </object>
</value>
```

The Invocation Context

The **InvocationContext** contains the current Item, the current ExoContainer and the current EventType is like below:

```
public interface InvocationContext extends ContextBase {

  Item getCurrentItem();

  int getEventType();

  ExoContainer getContainer();
}
```

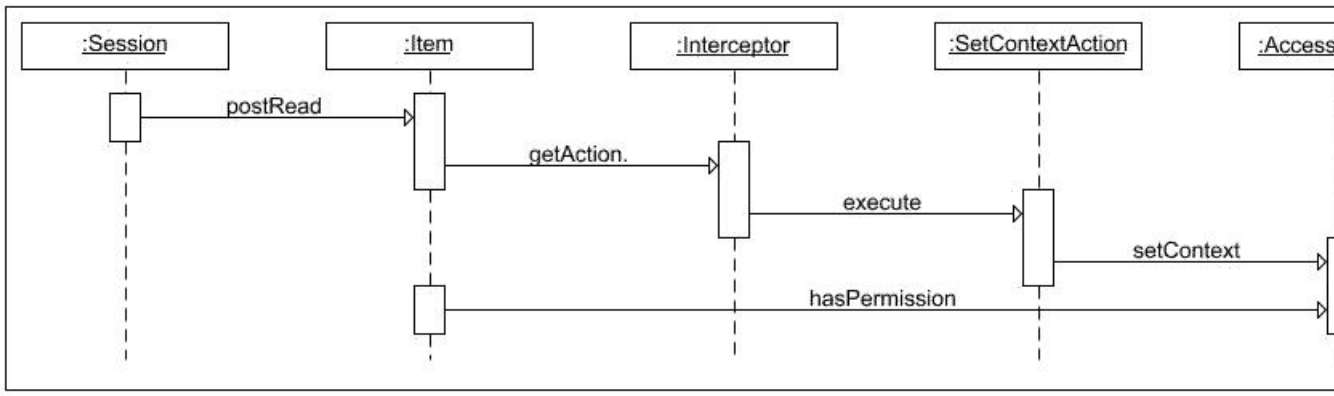
Custom Extended Access Manager

By default, all Workspaces share an AccessManager instance, created by RepositoryService at the startup (DefaultAccessManagerImpl) which supports default access control policy as described in the **Access Control** chapter. Custom Access Control policy can be applied to certain Workspace configuring **access-manager** element inside **workspace** as follows:

```
<workspace name="ws">
   ...
  <!-- after query-handler element -->
  <access-manager class="org.exoplatform.services.jcr.CustomAccessManagerImpl">
     <properties>
        <property name="someProperty" value="value"/>
        ...
     </properties>
  </access-manager>
  ...
</workspace>
```

When implementing AccessManager, hasPermission() method has to be overriden so it uses the current invocation context at its discretion. For instance, it may get the current node's metadata and make a decision if the current User has appropriate permissions. Use Invocation Context's runtime properties to make a decision about current Session's privileges (see the Example below)

Simplified Sequence diagram for the Session.getNode() method (as an Example):



Example of a custom Access Manager

The sample CustomAccessManagerImpl below extends the default access manager and uses some DecisionMakingService in the overloaded hasPermission method to find out if a current user has permission to use current **item, event type, userID** and some parameter of AccessManager. To make this Access manager work, it is necessary to configure it in jcr configuration as mentioned in **Custom Extended Access Manager** and SetAccessContextAction should be configured in the way mentioned in **Access Context Action.**

```
public class CustomAccessManagerImpl extends AccessManager {

 private String property;
 private DecisionMakingService theService;

 public CustomAccessManagerImpl (RepositoryEntry config, WorkspaceEntry wsConfig,
            OrganizationService orgService, DecisionMakingService someService) throws
RepositoryException {
  super(config, wsConfig, orgService);
  this.property = wsConfig.getAccessManager().getParameterValue("someParam");
  this.theService = someService;
 }

 @Override
 public boolean hasPermission(AccessControlList acl, String[] permission, String userId) {
  // call the default permission check
  if (super.hasPermission(acl, permission, userId)) {

   Item curItem = context().getCurrentItem();
   int eventType = context().getEventType();
   ExoContainer container = context().getContainer();
```

```
// call some service's method
      return theService.makeDecision(curItem, eventType, userId, property);
    } else {
      return false;
    }
  }
}
```

Link Producer Service

Link Producer Service - a simple service, which generates an .lnk file, that is compatible with the Microsoft link file format. It is an extension of the REST Framework library and is included into the WebDav service. On dispatching a GET request the service generates the content of an .lnk file, which points to a JCR resource via WebDav.

Link Producer has a simple configuration like described below:

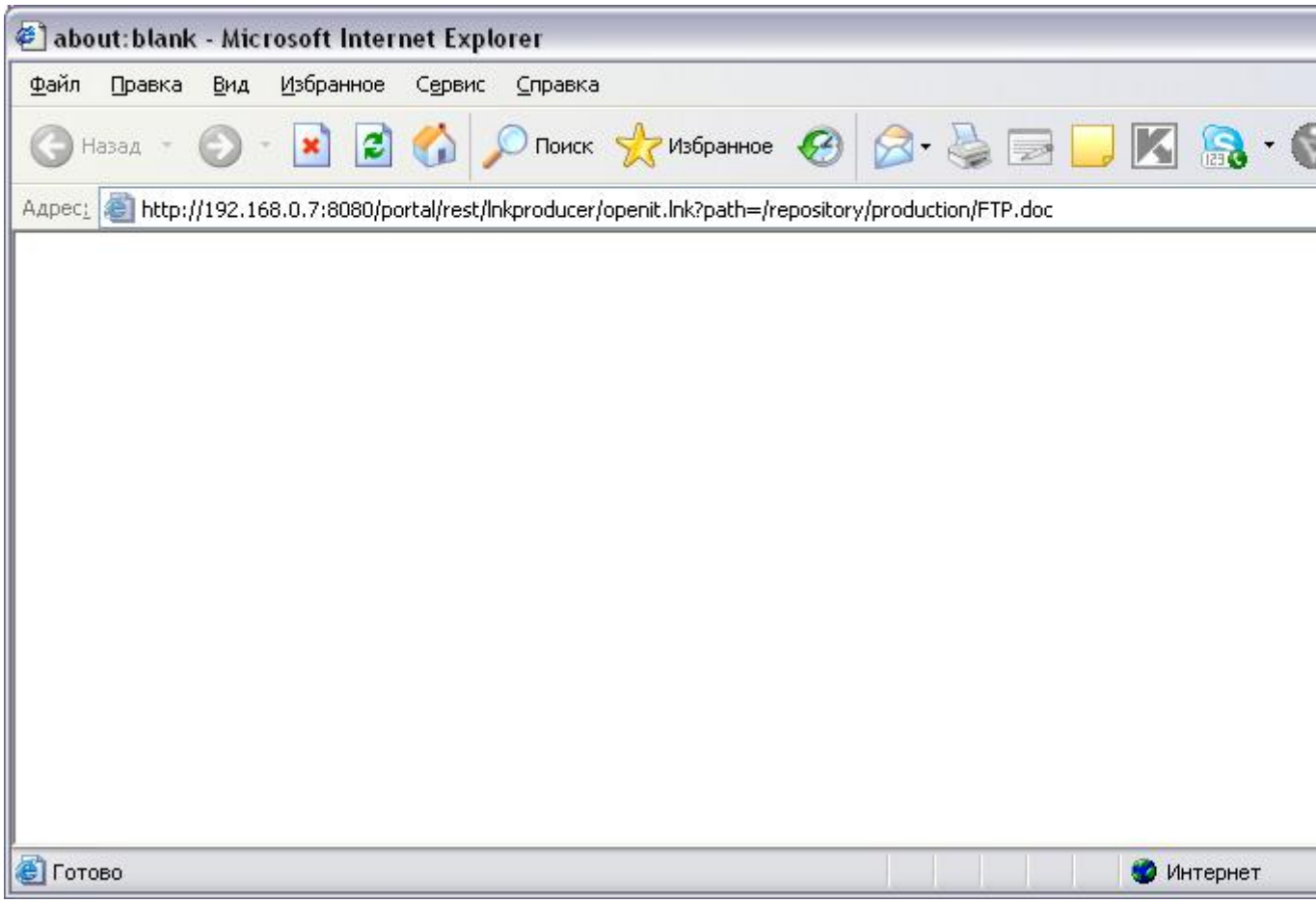
```
<component>
  <key>org.exoplatform.services.webdav.lnkproducer.LnkProducer</key>
  <type>org.exoplatform.services.webdav.lnkproducer.LnkProducer</type>
</component>
```

When using JCR the resource can be addressed by WebDav reference (href) like `http://host:port/rest/jcr/repository/workspace/somenode/somefile.extention` , the link servlet must be called for this resource by several hrefs, like `http://localhost:8080/rest/lnkproducer/openit.lnk?path=/repository/workspace/somenode/somefile.extention`

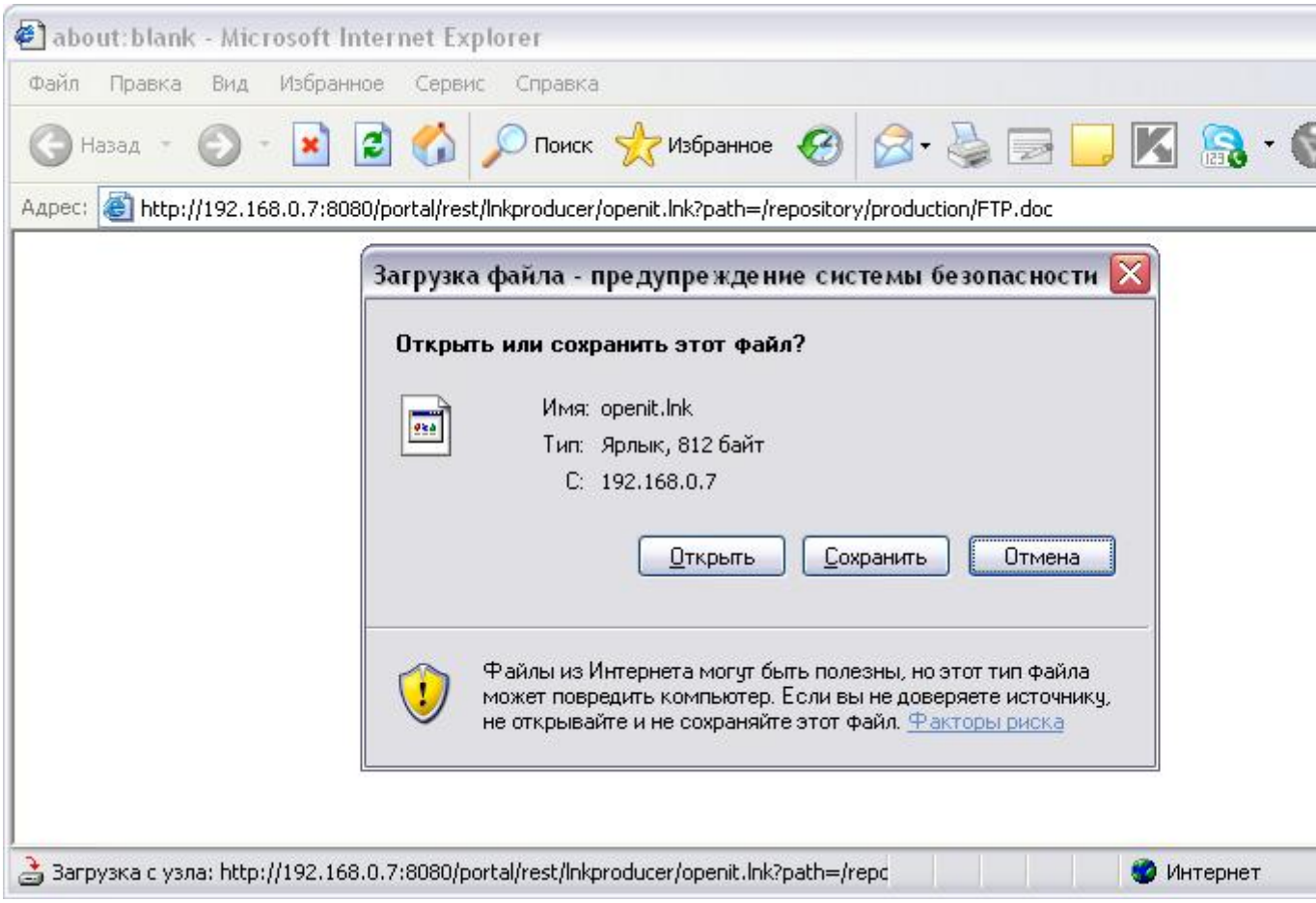
Please note, that when using the portal mode the REST servlet is available using a reference (href) like `http://localhost:8080/portal/rest/...`

The name of the .lnk file can be any. But for the best compatibility it must be the same as the name of the JCR resource.

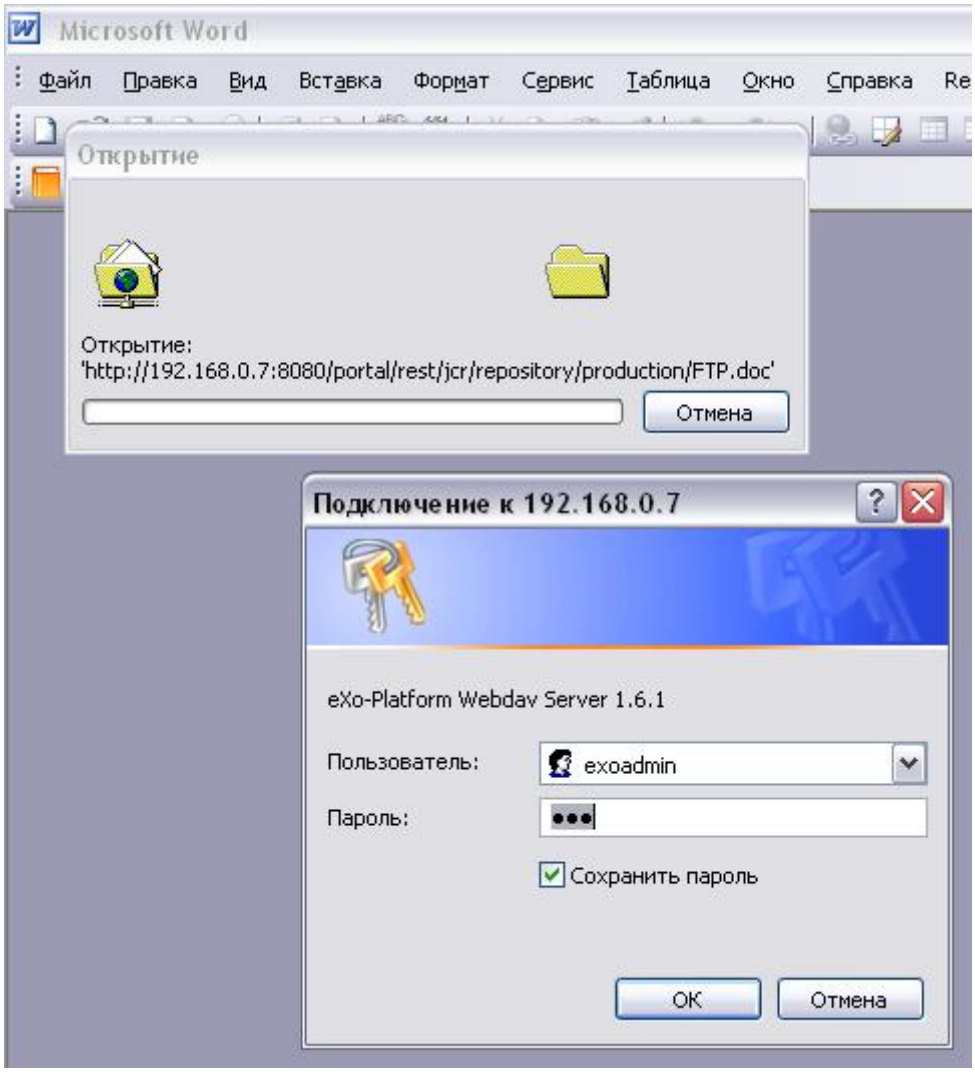
Here is a step by step sample of a use case of the link producer... At first, type valid reference to the resource, using the link producer in your browser's adress field:



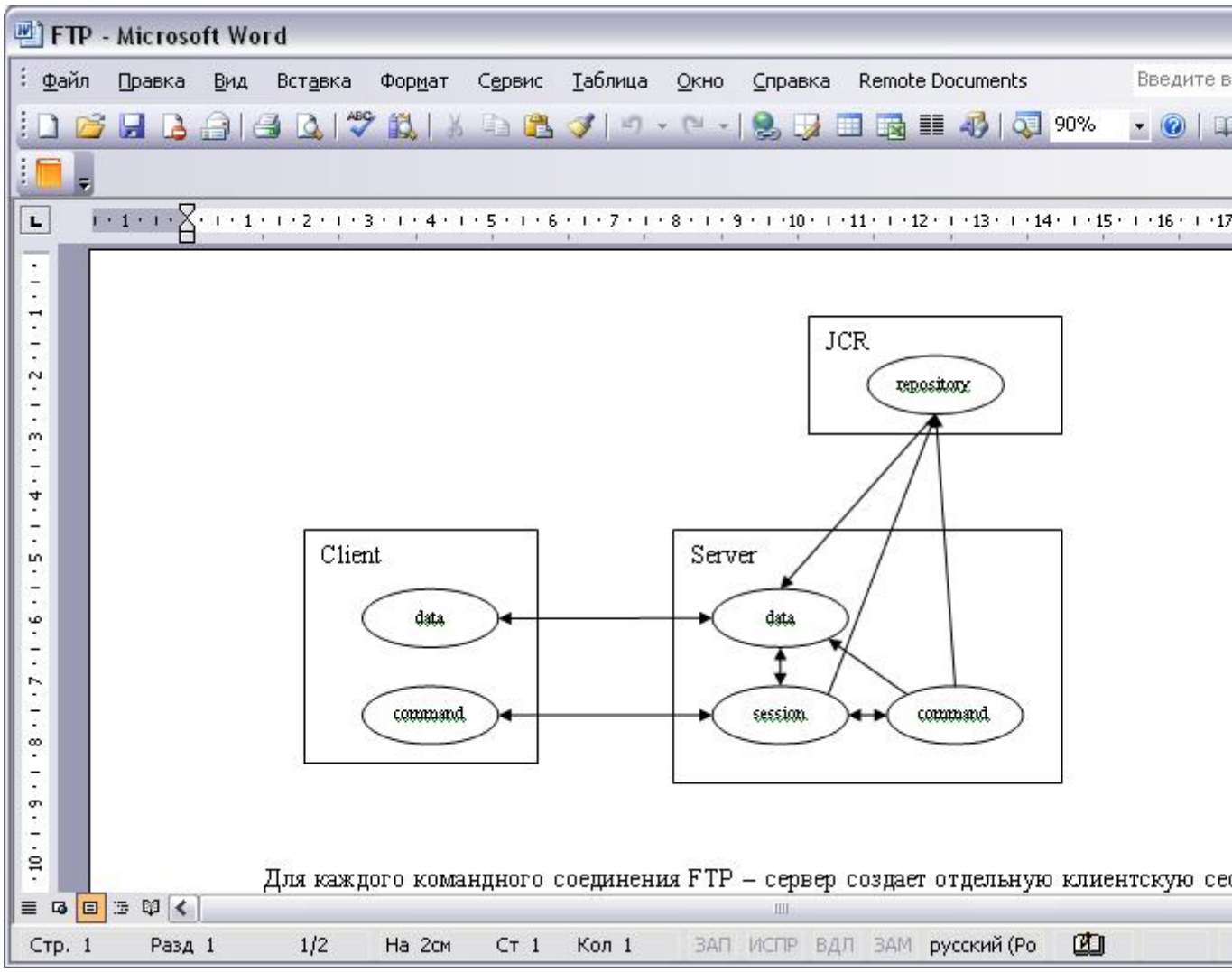
Internet Explorer will give a dialog window requesting to Open a file or to Save it. Click on the Open button



In Windows system an .lnk file will be downloaded and opened with the application which is registered to open the files, which are pointed to by the .lnk file. In case of a .doc file, Windows opens Microsoft Office Word which will try to open a remote file (test0000.doc). Maybe it will be necessary to enter USERNAME and PASSWORD.



Next, you will be able to edit the file in Microsoft Word.



The Link Producer is necessary for opening/editing and then saving the remote files in Microsoft Office Word, without any further updates.

Also the Link Producer can be referenced to from an HTML page. If page contains code like

```
<a href="http://localhost:8080/rest/lnkproducer/openit.lnk?path=/repository/workspace/somenode/somefile.extention">somefile.extention</a>
```

the file "somefile.extention" will open directly.

Binary Values Processing

Configuration

Binary large object (BLOB) properties can be stored in two ways in the eXo JCR: in the database with items information or in an external storage on host file system. These options can be configured at workspace in the repository configuration file (repository-configuration.xml in portal and exo-jcr-config.xml in standalone mode). The database storage can't be completely disabled.

The first case is optimal for most of cases which you do not use very large values or/and do not have too many BLOBs. The configuration of the BLOBs size and BLOBs quantity in a repository depend on your database features and hardware.

The second case is to use an external values storage. The storage can be located on a built-in hard disk or on an attached storage. But in any cases, you should access to the storage as if it was a regular file(s). The external value storage is optional and can be enabled in a database configuration.

Note

eXo JCR Repository service configuration basics is discussed in *Configuration*

Database and workspace persistence storage configuration is discussed in *JDBC Data Container config*

Configuration details for *External Value Storages*.

Usage

In both of the cases, a developer can set/update the binary Property via Node.setProperty(String, InputStream), Property.setValue(InputStream) as described in the spec JSR-170. Also, there is the setter with a ready Value object (obtainer from ValueFactory.createValue(InputStream)).

An example of a specification usage.

```
// Set the property value with given stream content.
Property binProp = node.setProperty("BinData", myDataStream);
// Get the property value stream.
InputStream binStream = binProp.getStream();

// You may change the binary property value with a new Stream, all data will be replaced
// with the content from the new stream.
Property updatedBinProp = node.setProperty("BinData", newDataStream);
// Or update an obtained property
updatedBinProp.setValue(newDataStream);
// Or update using a Value object
```

```
updatedBinProp.setValue(ValueFactory.createValue(newDataStream));
// Get the updated property value stream.
InputStream newStream = updatedBinProp.getStream();
```

But if you need to update the property sequentially and with partial content, you have no choice but to edit the whole data stream outside and get it back to the repository each time. In case of really large-sized data, the application will be stuck and the productivity will decrease a lot. JCR stream setters will also check constraints and perform common validation each time.

There is a feature of the eXo JCR extension that can be used for binary values partial writing without frequent session level calls. The main idea is to use a value object obtained from the property as the storage of the property content while writing/reading during runtime.

According to the spec JSR-170, Value interface provides the state of property that can't be changed (edited). The eXo JCR core provides ReadableBinaryValue and EditableBinaryValue interfaces which themselves extend JCR Value. The interfaces allow the user to partially read and change a value content.

ReadableBinaryValue value can be casted from any value, i.e. String, Binary, Date etc.

```
// get the property value of type PropertyType.STRING
ReadableBinaryValue extValue = (ReadableBinaryValue)
 node.getProperty("LargeText").getValue();
// read 200 bytes to a destStream from the position 1024 in the value content
OutputStream destStream = new FileOutputStream("MyTextFile.txt");
extValue.read(destStream, 200, 1024);
```

But EditableBinaryValue can be applied only to properties of type PropertyType.BINARY. In other cases, a cast to EditableBinaryValue will fail.

After the value has been edited, the EditableBinaryValue value can be applied to the property using the standard setters (Property.setValue(Value), Property.setValues(Value), Node.setProperty(String, Value) etc.). Only after the EditableBinaryValue has been set to the property, it can be obtained in this session by getters (Property.getValue(), Node.getProperty(String) etc.).

The user can obtain an EditableBinaryValue instance and fill it with data in an interaction manner (or any other appropriated to the targets) and return (set) the value to the property after the content will be done.

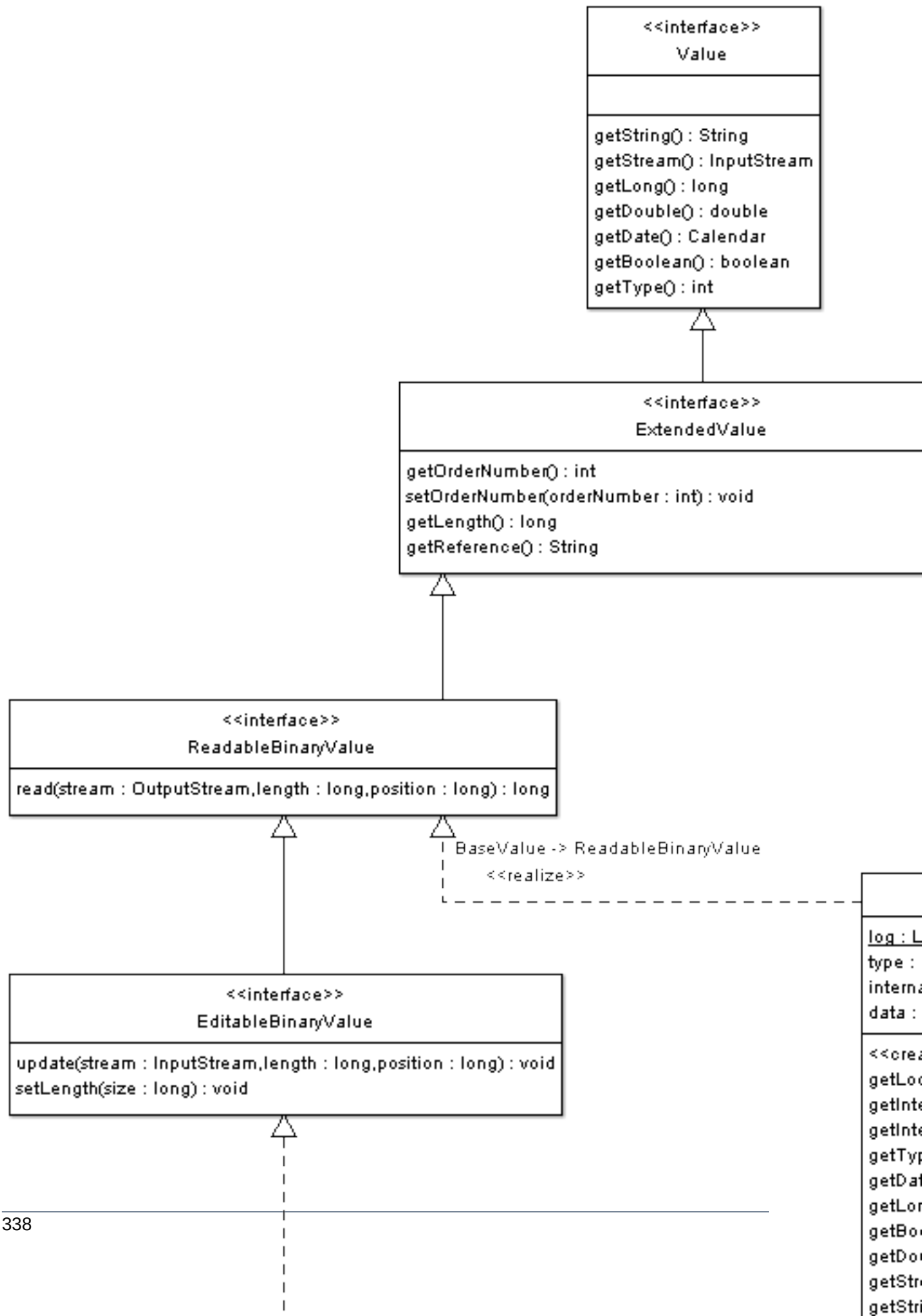
```
// get the property value for PropertyType.BINARY Property
EditableBinaryValue extValue = (EditableBinaryValue) node.getProperty("BinData").getValue();

// update length bytes from the stream starting from the position 1024 in existing Value data
```

```
extValue.update(dataInputStream, dataLength, 1024);

// apply the edited EditableBinaryValue to the Property
node.setProperty("BinData", extValue);

// save the Property to persistence
node.save();
```

A practical example of the iterative usage. In this example, the value is updated with data from the sequence of streams and after the update is done, the value will be applied to the property and be visible during the session.

```
// update length bytes from the stream starting from the particular
// position in the existing Value data
int dpos = 1024;
while (source.dataAvailable()) {
  extValue.update(source.getInputStream(), source.getLength(), dpos);
  dpos = dpos + source.getLength();
}

// apply the edited EditableBinaryValue to the Property
node.setProperty("BinData", extValue);
```

Value implementations



ReadableBinaryValue has one method to read Value.

Read length bytes is counted from the binary value to the given position into the stream.

```
long read(OutputStream stream, long length, long position) throws IOException, RepositoryException ;
```

EditableBinaryValue has two methods to edit value.

Update with length bytes from the specified stream to this value data at a position. If the position is lower than 0, the IOException exception will be thrown. If the position is higher than the current Value length, the Value length will be increased at first to the size of position and length bytes will be added after the position.

```
void update(InputStream stream, long length, long position) throws IOException;
```

Set the length of the Value in bytes to the specified size. If the size is lower than 0, the IOException exception will be thrown. This operation can be used to extend or truncat the Value size. This method is used internally in the update operation in case of extending the size to the given position.

```
void setLength(long size) throws IOException;
```

An application can perform JCR binary operations more flexibly and will have less I/O and CPU usage using these methods.

JCR Resources:

* Java Community Process: *JSR 170* [http://jcp.org/en/jsr/detail?id=170] and *JSR 283* [http://jcp.org/en/jsr/detail?id=283]

* Tom Wheeler, *The Java Content Repository* [http://www.tomwheeler.com/java_content_repository_tomwheeler_20071007.pdf] (2007)

* Roy T. Fielding, *JSR 170 Overview: Standardizing the Content Repository Interface* [http://www.day.com/content/dam/day/whitepapers/JSR_170_White_Paper.pdf] (March 13, 2005)

* David Nuescheler and Janus Boye, *JSR-170 What's in it for me?* [http://www.cmswatch.com/Feature/123] (April 20, 2005)

* Benjamin Mestrallet, Tuan Nguyen, Gennady Azarenkov, Francois Moron and Brice Revenant *eXo Platform v2, Portal, JCR, ECM, Groupware and Business Intelligence* [http://www.theserverside.com/tt/articles/article.tss?l=eXoPlatform2] (January 2006)

JCR Workspace Data Container (architecture contract)

Goals

- Cover the requirements on Workspace Data Container implementation
- Describe container life cycle
- Describe relations between container and high-level DataManagers

Concepts

Container and connection

Workspace Data Container (container) serves Repository Workspace persistent storage. WorkspacePersistentDataManager (data manager) uses container to perform CRUD operation on the persistent storage. Accessing to the storage in the data manager is implemented via storage connection obtained from the container (WorkspaceDataContainer interface implemenatiton). Each connection represents a transaction on the storage. Storage Connection (connection) should be an implementation of WorkspaceStorageConnection.

- Container acts as a factory of a new storage connections. Usually, this method is designed to be synchronized to avoid possible concurrent issues.

```
WorkspaceStorageConnection openConnection() throws RepositoryException;
```

- Open read-only WorkspaceStorageConnection. Read-only connections can be potentially a bit faster in some cases.

```
WorkspaceStorageConnection openConnection(boolean readOnly) throws RepositoryException;
```

EXPERIMENTAL

Read-only WorkspaceStorageConnection is experimental feature and not currently handled in JCR. Actually, such connections didn't prove their performance, so JCR Core doesn't use them.

- Storage connection might also be reused. This means reuse of physical resource (e.g. JDBC Connection) allocated by one connection in another. This feature is used in a data manager for saving ordinary and system changes on the system Workspace. But the reuse is an optional feature and it can work, otherwise a new connection will open.

```
WorkspaceStorageConnection reuseConnection(WorkspaceStorageConnection original) throws
 RepositoryException;
```

- When checking Same-Name Siblings (SNS) existence, JCR Core can use new connection or not. This is defined via Workspace Data Container configuration and retrieved by using a special method.

```
boolean isCheckSNSNewConnection();
```

Container initialization is only based on a configuration. After the container has been created, it's not possible to change parameters. Configuration consists of implementation class and set of properties and Value Storages configuration.

Value storages

Container provides optional special mechanism for Value storing. It's possible to configure external Value Storages via container configuration (available only via configuration). Value Storage works as fully independent pluggable storage. All required parameters storage obtains from its configuration. Some storages are possible for one container. Configuration describes such parameters as ValueStoragePluginimplementation class, set of implementation specific properties and filters. The filters declares criteria for Value matching to the storage. Only matched Property Values will be stored. So, in common case, the storage might contains only the part of the Workspace content. Value Storages are very useful for BLOB storing. E.g. storing on the File System instead of a database.

Container obtains Values Storages from ValueStoragePluginProvider component. Provider acts as a factory of Value channels (ValueIOChannel). Channel provides all CRUD operation for Value Storage respecting the transaction manner of work (how it can be possible due to implementation specifics of the storages).

Lifecycle

Container is used for read and write operations by data manager. Read operations (getters) uses connection once and close it on the finally. Write operations performs in commit method as a sequence of creating/ updating calls and final commit (or rollback on error). Writes uses one connection (or two - another for system workspace) per commit call. One connection guaranties transaction support for write operations. Commit or rollback should free/clean all resources consumed by the container (connection).

Value storage lifecycle

Value storage is used from the container inside. Reads are related to a container reads. Writes are commit-related. Container (connection) implementation should use transaction capabilities of the storages in the same way as for other operations.

Requirements

Connection creation and reuse should be a thread safe operation. Connection provides CRUD operations support on the storage.

Read operations

- Read ItemData from the storage by item identifier.

```
ItemData getItemData(String identifier) throws RepositoryException, IllegalStateException;
```

- Read ItemData from the storage by using the parent and name of the item, related to the parent location.

```
ItemData getItemData(NodeData parentData, QPathEntry name) throws RepositoryException,IllegalStateException;
```

- Read List of NodeData from the storage by using the parent location of the item.

```
List<NodeData> getChildNodesData(NodeData parent) throws RepositoryException, IllegalStateException;
```

- Reads List of PropertyData from the storage by using the parent location of the item.

```
List<PropertyData> getChildPropertiesData(NodeData parent) throws RepositoryException, IllegalStateException;
```

- Reads List of PropertyData with empty ValueData from the storage by using the parent location of the item.

This methiod specially dedicated for non-content modification operations (e.g. Items delete).

```
List<PropertyData> listChildPropertiesData(NodeData parent) throws RepositoryException, IllegalStateException;
```

- Reads List of PropertyData from the storage by using the parent location of the item.

It's REFERENCE type: Properties referencing Node with given nodeIdentifier. See more in javax.jcr.Node.getReferences()

```
List<PropertyData> getReferencesData(String nodeIdentifier) throws RepositoryException,IllegalStateException,UnsupportedOperationException;
```

Write operations

- Add single NodeData.

```
void add(NodeData data) throws RepositoryException,UnsupportedOperationException,InvalidItemStateException,IllegalStateException;
```

- Add single PropertyData.

```
void add(PropertyData data) throws RepositoryException,UnsupportedOperationException,InvalidItemStateException,IllegalStateException;
```

- Update NodeData.

```
void update(NodeData data) throws RepositoryException,UnsupportedOperationException,InvalidItemStateException,IllegalStateException;
```

- Update PropertyData.

```
void update(PropertyData data) throws RepositoryException,UnsupportedOperationException,InvalidItemStateException,IllegalStateException;
```

- Rename NodeData by using Node identifier and new name and indexing from the data.

```
void rename(NodeData data) throws RepositoryException,UnsupportedOperationException,InvalidItemStateException,IllegalStateException;
```

- Delete NodeData.

```
void delete(NodeData data) throws RepositoryException,UnsupportedOperationException,InvalidItemStateException,IllegalStateException;
```

- Delete PropertyData.

```
void delete(PropertyData data) throws RepositoryException,UnsupportedOperationException,InvalidItemStateException,IllegalStateException;
```

- Persist changes and closes connection. It can be database transaction commit for instance etc.

```
void commit() throws IllegalStateException, RepositoryException;
```

- Refuse persistent changes and closes connection. It can be database transaction rollback for instance etc.

```
void rollback() throws IllegalStateException, RepositoryException;
```

All methods throw IllegalStateException if connection is closed. UnsupportedOperationException if the method is not supported (e.g. JCR Level 1 implementation etc). RepositoryException if some errors occur during preparation, validation or persistence.

State operations

- Return true if connection can be used.

```
boolean isOpened();
```

Validation of write operations

Container has to care about storage consistency (JCR constraints) on write operations: (InvalidItemStateException should be thrown according the spec). At least, the following checks should be performed:

- On ADD errors
  - Parent not found. Condition: Parent ID (Item with ID is not exists).
  - Item already exists. Condition: ID (Item with ID already exists).
  - Item already exists. Condition: Parent ID, Name, Index (Item with parent ID, name and index already exists).
- On DELETE errors
  - Item not found. Condition ID.
  - Can not delete parent till children exists.
- On UPDATE errors
  - Item not found. Condition ID.
  - Item already exists with higher Version. Condition: ID, Version (Some Session had updated Item with ID prior this update).

Consistency of save

The container (connection) should implement consistency of Commit (Rollback) in **transaction manner**. I.e. If a set of operations was performed **before** the future **Commit** and another next operation **fails**. **It should be possible to** rollback applied changes using **Rollback** command.

Value storages API

Storages provider:

Container implementation obtains Values Storages option via ValueStoragePluginProvider component. Provider acts as a factory of Value channels (ValueIOChannel) and has two methods for this purpose:

- Return ValueIOChannel matched this property and valueOrderNumer. Null will be returned if no channel matches.

```
ValueIOChannel getApplicableChannel(PropertyData property, int valueOrderNumer) throws IOException;
```

- Return ValueIOChannel associated with given storageId.

```
ValueIOChannel getChannel(String storageId) throws IOException, ValueStorageNotFoundException;
```

There is also method for consistency check, but this method doesn't used anywhere and storage implementations has it empty.

Value storage plugin

Provider implementation should use ValueStoragePlugin abstract class as a base for all storage implementations. Plugin provides support for provider implementation methods. Plugin's methods should be implemented:

- Initialize this plugin. Used at start time in ValueStoragePluginProvider.

```
public abstract void init(Properties props, ValueDataResourceHolder resources) throws RepositoryConfigurationException, IOException;
```

- Open ValueIOChannel.Used in ValueStoragePluginProvider.getApplicableChannel(PropertyData, int) and getChannel(String)

```
public abstract ValueIOChannel openIOChannel() throws IOException;
```

- Return true if this storage has the same storageId.

```
public abstract boolean isSame(String valueDataDescriptor);
```

Value I/O channel

Channel should implement ValueIOChannel interface. CRUD operation for Value Storage:

- Read Property value.

```
ValueData read(String propertyId, int orderNumber, int maxBufferSize) throws IOException;
```

- Add or update Property value.

```
void write(String propertyId, ValueData data) throws IOException;
```

- Delete Property all values.

```
void delete(String propertyId) throws IOException;
```

Transaction support via channel

Modification operations should be applied only when commiting. Rollback is required for data created cleanup.

- Commit channel changes.

```
void commit() throws IOException;
```

- Rollback channel changes.

```
void rollback() throws IOException;
```

How-to implement Workspace Data Container

Short intro about Workspace data container implementation practices:

1. Read a bit about the *contract*.
2. Start new implementation project pom.xml with org.exoplatform.jcr parent. (optional, but will makes the development easy)
3. Update sources of JCR Core and read JavaDoc on **org.exoplatform.services.jcr.storage.WorkspaceDataContainer** and **org.exoplatform.services.jcr.storage.WorkspaceStorageConnection** interfaces. They are the main part for the implemenation.
4. Look at **org.exoplatform.services.jcr.impl.dataflow.persistent.WorkspacePersistentDataManager** sourcecode, check how data menager uses container and its connections (see in save() method)
5. Create **WorkspaceStorageConnection** dummy implementation class. It's freeform class, but to be close to the eXo JCR, check how to implement JDBC or SimpleDB containers (**org.exoplatform.services.jcr.impl.storage.jdbc.JDBCStorageConnection** and **org.exoplatform.services.jcr.aws.storage.sdb.SDBWorkspaceStorageConnection**). Take in account usage of **ValueStoragePluginProvider** in both implementations.Value storage is an useful option for production versions. But leave it to the end of implementation work.
6. Create the connection implementation unit tests to play TTD. (optional, but takes many benefits for the process)
7. Implement CRUD starting from the read to write etc. Test the methods by using the external implementation ways of data read/write in your backend.
8. When all methods of the connection done start **WorkspaceDataContainer**. Container class is very simple, it's like a factory for the connections only.
9. Care about container reuseConnection(WorkspaceStorageConnection) method logic. For some backends, it cab be same as openConnection(), but for some others, it's important to reuse physical backend connection, e.g. to be in the same transaction - see JDBC container.
10. It's almost ready to use in data manager. Start another test and go on.

When the container will be ready to run as JCR persistence storage (e.g. for this level testing), it should be configured in Repository configuration.

Assuming that our new implementation class name is **org.project.jcr.impl.storage.MyWorkspaceDataContainer**.

```
 <repository-service default-repository="repository">
 <repositories>
                <repository name="repository" system-workspace="production" default-
workspace="production">
    .............
    <workspaces>
     <workspace name="production">
      <container class="org.project.jcr.impl.storage.MyWorkspaceDataContainer">
       <properties>
        <property name="propertyName1" value="propertyValue1" />
        <property name="propertyName2" value="propertyValue2" />
        .......
        <property name="propertyNameN" value="propertyValueN" />
       </properties>
       <value-storages>
        .......
       </value-storages>
      </container>
```

Container can be configured by using set properties.

Notes on Value storage usage:

Value storages are pluggable to the container but if they are used, the container implementation should respect set of interfaces and external storage usage principles.

If the container has **ValueStoragePluginProvider** (e.g. via constructor), it's just a few methods to manipulate external Values data.

```
// get channel for ValueData write (add or update)
ValueIOChannel channel = valueStorageProvider.getApplicableChannel(data,  i);
if (channel == null) {
  // write
  channel.write(data.getIdentifier(),  vd);
  // obtain storage id,  id can be used for linkage of external ValueData and PropertyData in main
 backend
  String storageId = channel.getStorageId();
```

```
}

....

// delete all Property Values in external storage
ValueIOChannel channel = valueStorageProvider.getChannel(storageId);
channel.delete(propertyData.getIdentifier());

....

// read ValueData from external storage
ValueIOChannel channel = valueStorageProvider.getChannel(storageId);
ValueData vdata = channel.read(propertyData.getIdentifier(),  orderNumber,  maxBufferSize);
```

After a sequence of write and/or delete operations on the storage channel, the channel should be committed (or rolled back on an error). See **ValueIOChannel.commit()** and **ValueIOChannel.rollback()** and how those methods are used in JDBC container.

DBCleanService

API

It is special service for removing data from database.

Note

Code that invokes methods of DBCleanService must have JCRRuntimePermissions.MANAGE_REPOSITORY_PERMISSION permission;

There are three methods of DBCleanerService:

Table 47.1. API

public static void cleanWorkspaceData(WorkspaceEntry wsEntry)	Clean workspace data from database. Tables will be removed in case of multiDB, and only records will be removed in case of singleDB.
public static void cleanRepositoryData(RepositoryEntry repoEntry)	Cleanup repository data from database.
public static getDBCleaner(Connection jdbcConn, WorkspaceEntry wsEntry)	Returns DBClean object with defined connection that allow to manual invoke clean method on it. Note: DBClean doesn't perform commit or close connection. It should be done manually.

JCR Performance Tuning Guide

Introduction

This guide will show you possible ways of improving JCR performance.

It is intended to GateIn Administrators and those who wants to use JCR features.

JCR Performance and Scalability

Cluster configuration

EC2 network: 1Gbit

Servers hardware:

7.5 GB memory
4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each)
850 GB instance storage (2×420 GB plus 10 GB root partition)
64-bit platform
I/O Performance: High
API name: m1.large

Note

NFS and statistics (cacti snmp) server were located on one physical server.

JBoss AS configuration

```
JAVA_OPTS:         -Dprogram.name=run.sh         -server         -Xms4g         -
Xmx4g       -XX:MaxPermSize=512m       -Dorg.jboss.resolver.warning=true       -
Dsun.rmi.dgc.client.gcInterval=3600000  -Dsun.rmi.dgc.server.gcInterval=3600000
-XX:+UseParallelGC -Djava.net.preferIPv4Stack=true
```

JCR Clustered Performance

Benchmark test using webdav (Complex read/write load test (benchmark)) with 20K same file. To obtain per-operation results we have used custom output from the testscase threads to CSV file.

Read operation:

Warm-up iterations: 100
Run iterations: 2000
Background writing threads: 25
Reading threads: 225

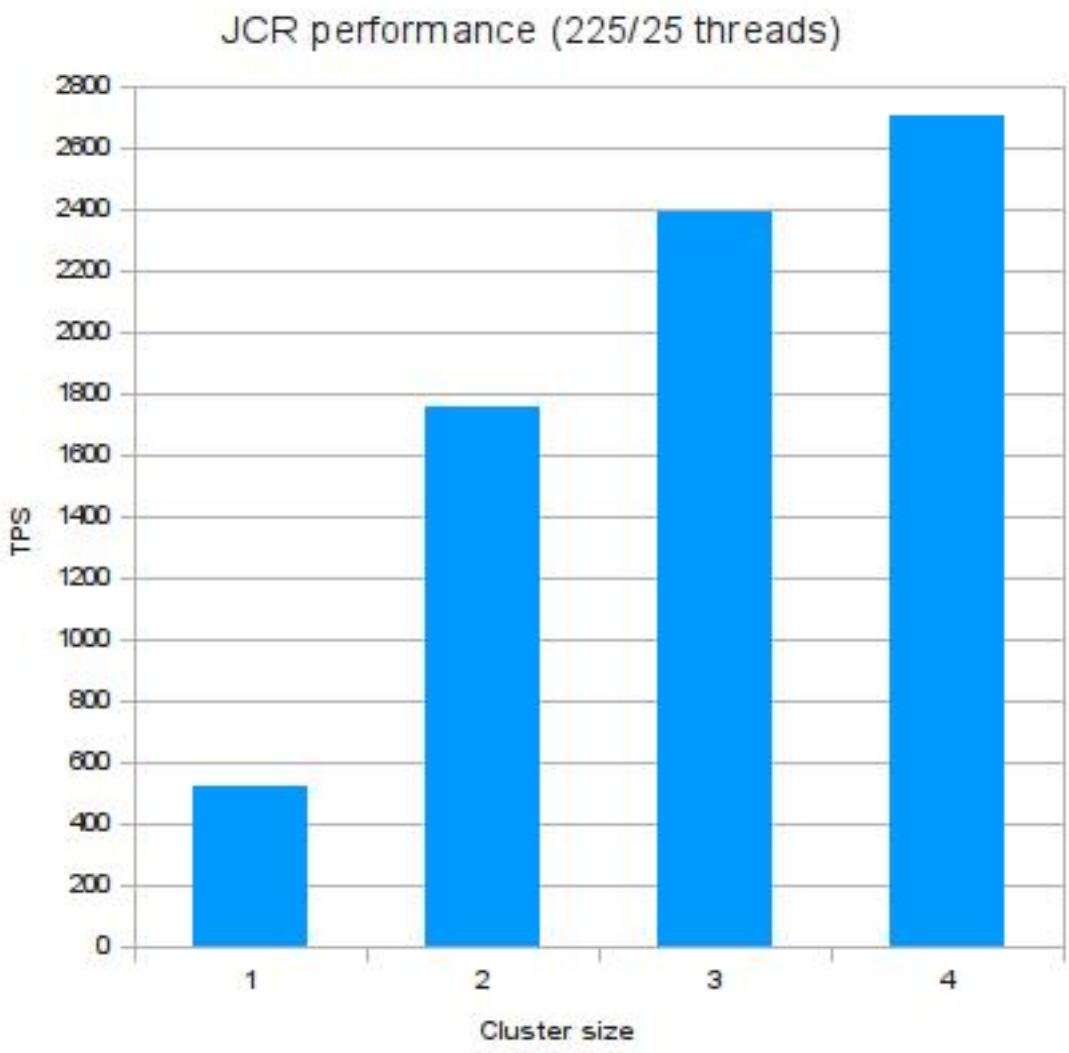


Table 48.1.

Nodes count	tps	Responses >2s	Responses >4s
1	523	6.87%	1.27%
2	1754	0.64%	0.08%
3	2388	0.49%	0.09%
4	2706	0.46%	0.1%

Read operaion with more threads:

Warm-up iterations: 100
Run iterations: 2000
Background writing threads: 50
Reading threads: 450

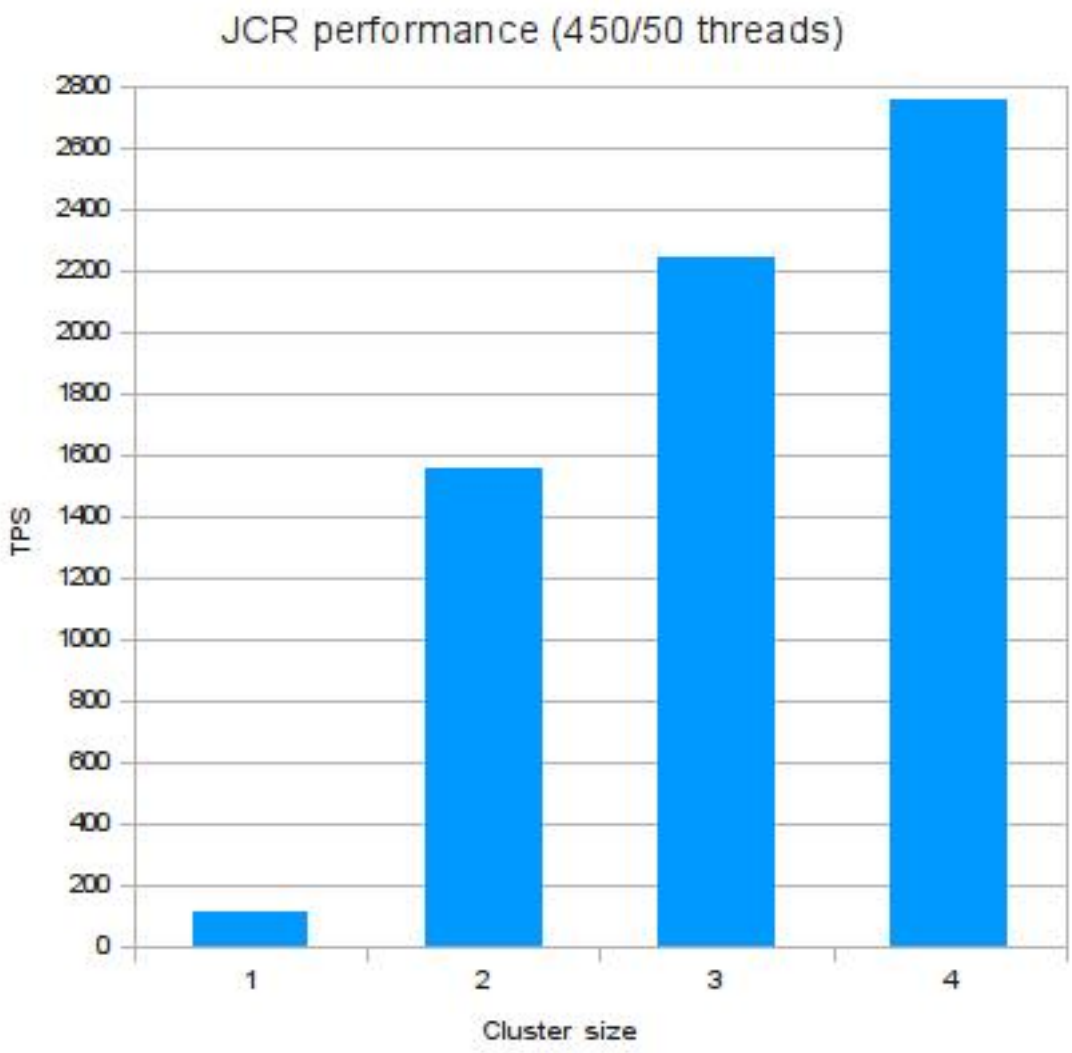


Table 48.2.

Nodes count	tps	Responses >2s	Responses >4s
1	116	?	?
2	1558	6.1%	0.6%
3	2242	3.1%	0.38%
4	2756	2.2%	0.41%

Performance Tuning Guide

JBoss AS Tuning

You can use *maxThreads* parameter to increase maximum amount of threads that can be launched in AS instance. This can improve performance if you need a high level of concurrency. also you can use `-XX:+UseParallelGC` java directory to use paralel garbage collector.

Beware of setting *maxThreads* too big, this can cause `OutOfMemoryError`. We've got it with `maxThreads=1250` on such machine:

7.5 GB memory
4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each)
850 GB instance storage (2×420 GB plus 10 GB root partition)
64-bit platform
I/O Performance: High
API name: m1.large
java -Xmx 4g

JCR Cache Tuning

Cache size

JCR-cluster implementation is built using JBoss Cache as distributed, replicated cache. But there is one particularity related to remove action in it. Speed of this operation depends on the actual size of cache. As many nodes are currently in cache as much time is needed to remove one particular node (subtree) from it.

Eviction

Manipulations with eviction *wakeUpInterval* value doestn't affect on performance. Performance results with values from 500 up to 3000 are approximately equal.

Transaction Timeout

Using short timeout for long transactions such as Export/Import, removing huge subtree defined timeout may cause `TransactionTimeoutException`. [TODO] put recomended timeout value

Clustering

For performance it is better to have loadbalacer, DB server and shared NFS on different computers. If in some reasons you see that one node gets more load than others you can decrease this load using load value in load balancer.

JGroups configuration

It's recommended to use "multiplexer stack" feature present in JGroups. It is set by default in eXo JCR and offers higher performance in cluster, using less network connections also. If there are two or more clusters in your network, please check that they use different ports and different cluster names.

Write performance in cluster

Exo JCR implementation uses Lucene indexing engine to provide search capabilities. But Lucene brings some limitations for write operations: it can perform indexing only in one thread. Thats

why write performance in cluster is not higher than in singleton environment. Data is indexed on coordinator node, so increasing write-load on cluster may lead to ReplicationTimeout exception. It occurs because writing threads queue in the indexer and under high load timeout for replication to coordinator will be exceeded.

Taking in consideration this fact, it is recommended to exceed `replTimeout` value in cache configurations in case of high write-load.

Replication timeout

Some operations may take too much time. So if you get `ReplicationTimeoutException` try increasing replication timeout:

```
<clustering mode="replication" clusterName="${jbosscache-cluster-name}">
  ...
  <sync replTimeout="60000" />
</clustering>
```

value is set in miliseconds.

JVM parameters

PermGen space size

If you intend to use Infinispan, you will have to increase the PermGen size to at least 256 Mo due to the latest versions of JGroups that are needed by Infinispan (please note that Infinspan is only dedicated to the community for now, no support will be provided). In case, you intend to use JBoss Cache, you can keep on using JGroups 2.6.13.GA which means that you don't need to increase the PermGen size.

Part II. eXoKernel

eXo Kernel

eXo Kernel introduction

eXo Kernel is the basis of all eXo platform products and modules. Any component available in eXo Platform is managed by the Exo Container, our micro container responsible for gluing the services through dependency injection

Therefore, each product is composed of a set of services and plugins registered to the container and configured by XML configuration files.

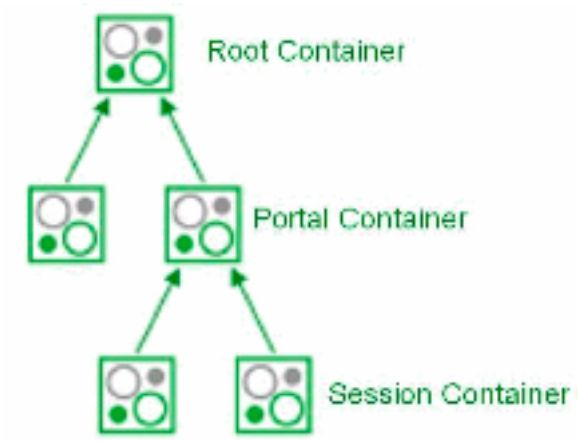
The Kernel module also contains a set of very low level services.

ExoContainer info

ExoContainer is the main IoC kernel object. The container is responsible for loading services/components.

Container hierarchy

- Behavior is like class loaders
- A child container sees parent container components
- Extensively used in eXo Platform



Service Configuration for Beginners

Related documents

- *Service Configuration in Detail*
- *Services Wiring*
- *Container Configuration*

Objective

We are going to talk about service configuration. You will learn about modes, services and containers, you will find out where the service configuration files have to be placed and you will also see the overriding mechanism of configurations. Finally you will understand how the container creates the services one after the other and what *Inversion of Control* really means.

Requirements

By reading this article you are already glancing at the heart of eXo Kernel.

Even you will read in this article to open the directory "exo-tomcat", you may have installed eXo Portal on any application server, just replace "exo-tomcat" by your folder name.

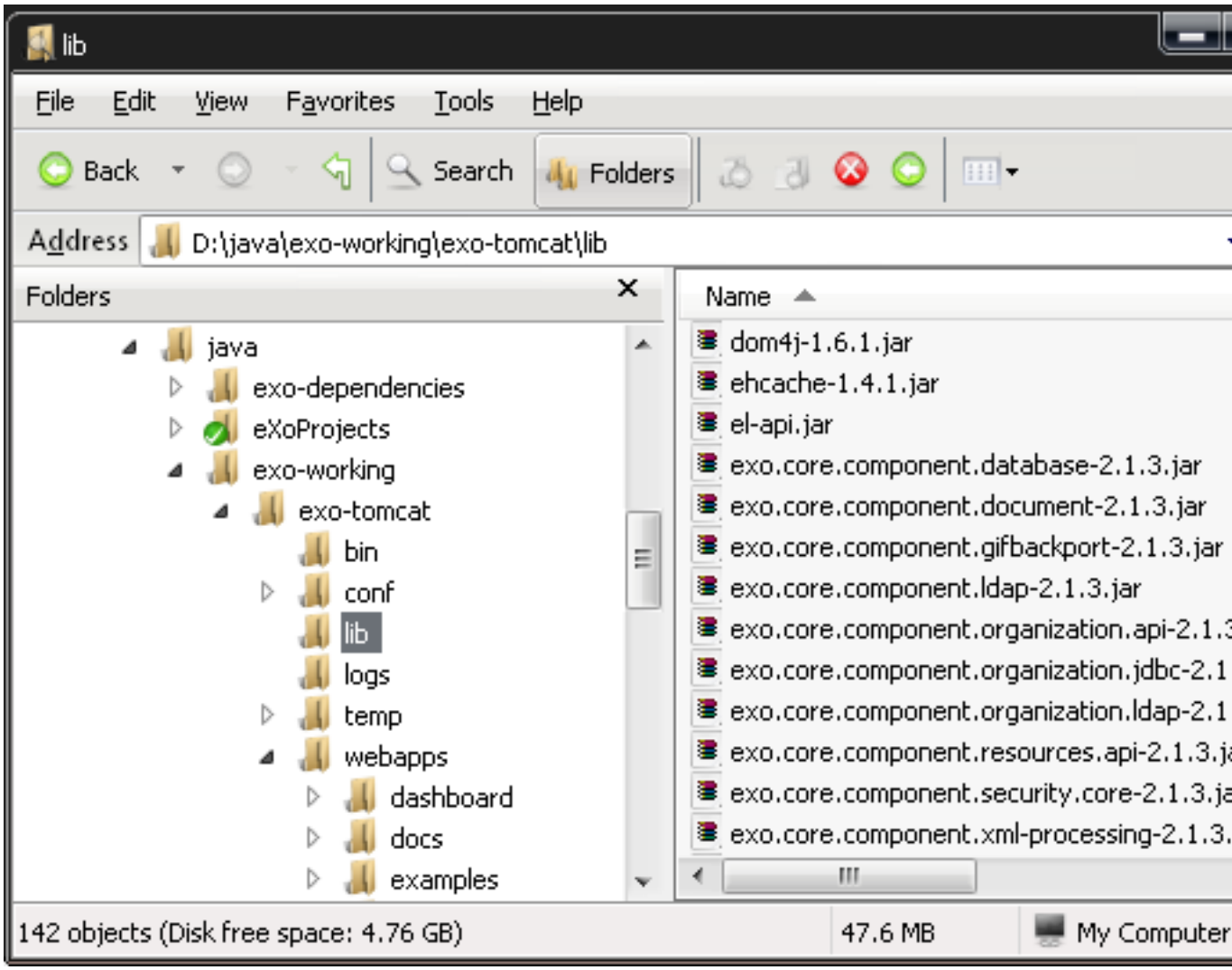
Note

If you only installed the all-in-one package for the eXo Portal, the folder paths are a slightly different. You have to replace *exo-tomcat* by *exo-eXoPortal-2.5.1-tomcat* (obviously depending on your version). Furthermore the webapps are delivered as war files.

You certainly already discovered eXo's fisheye URL (eXo is open source!) - *https://anonsvn.jboss.org/repos/exo-jcr/* - which allows you to surf in the source code of all classes, if you wish to do so.

Services

Nearly everything could be considered a service! To get a better idea, let's look into the *exo-tomcat/lib* folder where you find all deployed jar files.



For example you find services for databases, caching, ldap and ftp:

- exo.core.component.database-2.1.3.jar
- exo.kernel.component.cache-2.0.5.jar
- exo.core.component.organization.ldap-2.1.3.jar
- exo.jcr.component.ftp-1.10.1.jar

Of course, there are many more services, in fact a lot of these jar files are services. To find out you have to open the jar file and then look into its */conf* or */conf/portal* directory. Only if there is a file named *configuration.xml*, you are sure to have found a service.

Note

Why are there 2 different places to look for the configuration.xml? Because the */conf* directory is used by the `RootContainer` and the */conf/portal* directory is used by the `PortalContainer`. Later you will see more details about these containers.

Interface - Implementation It's important to get the idea that you separate the interface and implementation for a service. That is a good concept to reduce dependencies on specific implementations. This concept is well known for JDBC. If you use standard JDBC (=interface), you can connect any database (=implementation) to your application. In a similar way any service in eXo is defined by a java interface and may have many different implementations. The service implementation is then *injected* by a *container* into the application.

Singleton Each service has to be implemented as a *singleton* [http://en.wikipedia.org/wiki/Singleton_pattern], which means that each service is created only once - in one single instance.

Service = Component You always read about services, and you imagine a service as a large application which does big things, but that's not true, a service can be just a little *component* that reads or transforms a document, therefore the term component is often used instead of service - so bear in mind: *a service and a component can safely be considered to be the same thing*.

Configuration File

The jar file of a service should contain a default configuration, you find this configuration in the configuration.xml file which comes with the jar. A configuration file can specify several services, as well as there can be several services in one jar file.

For example open the exo.kernel.component.cache-2.0.5.jar file and inside this jar open /conf/portal/configuration.xml. You will see:

```
<component>
<key>org.exoplatform.services.cache.CacheService</key>
<type>org.exoplatform.services.cache.impl.CacheServiceImpl</type>
...
```

Here you will note that a service is specified between the `<component>` tags. Each service has got a key, which defines the kind of service. As you imagine the content of the `<key>` tag matches the *qualified java interface name* (`org.exoplatform.services.cache.CacheService`) of the service. The specific implementation class of the `CacheService` is defined in the `<type>` tag.

Parameters You have already opened some configuration files and seen that there are more than just `<key>` and `<type>` tags. You can provide your service with init parameters. The parameters can be simple parameters, properties, or object-params. There are also *plugins* and they are special because the container calls the setters of your service in order to *inject* your plugin in your service (called *setter injection*) see *Service Configuration in Detail*. In general your service is free to use init parameters, they are not required.

If you ever need to create your own service, the minimum is to create an empty interface, an empty class and a constructor for your class - that's all. Ok, you also should put your class and the interface in a jar file and add a default configuration file.

Execution Modes

One important thing to understand concerns execution modes. There are only two modes:

- Portal mode: The service runs embedded in the eXo Portal. In this mode a `PortalContainer` is used.
- Standalone mode: The service runs without the portal. For example, the JCR service can run standalone, and also the eXo Portlet Container. This mode is used by eXo developers for unit tests. As the name suggests a `StandaloneContainer` is used.

Containers

In order to access to a service you need to use a Container. Just open *https://anonsvn.jboss.org/repos/exo-jcr/kernel/trunk/exo.kernel.container/src/main/java/org/exoplatform/container*.

Among the classes you see in this directory, you only will be interested in these three container types:

- RootContainer: This is a base container. This container plays an important role during startup, but you should not use it directly.
- PortalContainer: Created at the startup of the portal web application (in the init() method of the PortalController servlet)
- StandaloneContainer: A context independent eXo Container. The `StandaloneContainer` is also used for unit tests.

Use only one container Even if there are several container types you always use exactly one. The RootContainer is never directly used and it depends on the execution mode if you use the PortalContainer or the StandaloneContainer. You will ask how to find out the execution mode in my application and how to manage these two modes. It's easy, you don't have to worry about it because the ExoContainerContext class provides a static method that allows you to get the right container from anywhere (see info box).

PicoContainer All containers inherit from the ExoContainer class which itself inherits from a `PicoContainer`. *PicoContainer* [http://www.picocontainer.org/] is a framework which allows eXo to apply the IoC (*Inversion of Control*) principles. The precise implementations of any service is unknown at compile time. Various implementations can be used, eXo supplies different implementations but they also may be delivered by other vendors. The decision which service to use during runtime is made in configuration files.

These configuration files are read by the container, the container adds all services to a list or more exactly a java HashTable. It's completely correct to suppose that the configuration.xml you already saw plays an important role. But there are more places where a configuration for a service can be defined as you see in the next chapter.

Note

"In your java code you have to use

```
ExoContainer myContainer = ExoContainerContext.getCurrentContainer();
```

in order to access to the current container. It doesn't greatly matter to your application if the current container is a `PortalContainer` or a `StandaloneContainer`. Once you have your container you may access to any service registered in this container using

```
MyService myService = (MyService) myContainer.getComponentInstance(MyService.class);
```

You easily realize that `MyService.class` is the name of the service interface.

Configuration Retrieval

The configuration you find inside the jar file is considered as the default configuration. If you want to override this default configuration you can do it in different places outside the jar. When the container finds several configurations for the same service, the configuration which is found later replaces completely the one found previously. Let's call this the *configuration override mechanism*.

RootContainer

As both containers, PortalContainer and StandaloneContainer, depend on the RootContainer, we will start by looking into this one.

The retrieval sequence in short:

1. Services default `RootContainer` configurations from JAR files */conf/configuration.xml*
2. External `RootContainer` configuration, to be found at *exo-tomcat/exo-conf/configuration.xml*

Note

Naturally you always have to replace *`exo-tomcat`* by your own folder name. In case of a Java Standalone application you have to use the *`user.dir`* JVM system property value.

HashTable The `RootContainer` creates a java `HashTable` which contains key-value pairs for the services. The qualified interface name of each service is used as key for the hashtable. Hopefully you still remember that the *<key>* tag of the configuration file contains the interface name? The value of each hashtable pair is an object that contains the service configuration (yes, this means the whole structure between the *<component>* tags of your `configuration.xml` file).

The `RootContainer` runs over all jar files you find in *exo-tomcat/lib* and looks if there is a configuration file at */conf/configuration.xml*, the services configured in this file are added to the hashtable. That way - at the end of this process - the default configurations for all services are stored in the hashtable.

Note

What happens if the same service - recognized by the same qualified interface name - is configured in different jars? As the service only can exist one time the configuration of the jar found later overrides the previous configuration. You know that the loading **order of the jars is unpredictable** you **must not depend on this**.

If you wish to provide your own configurations for one or several services, you can do it in a general configuration file that has to be placed at *exo-tomcat/exo-conf/configuration.xml*. Do not search for such a file on your computer - you won't find one, because this option is not used in the default installation. Here again the same rule applies: *The posterior configuration replaces the previous one*.

The further configuration retrieval depends on the container type.

PortalContainer

The PortalContainer takes the hashtable filled by the RootContainer and continues to look in some more places. Here you get the opportunity to replace RootContainer configurations by those which are specific to your portal. Again, the configurations are overridden whenever necessary.

In short PortalContainer configurations are retrieved in the following lookup sequence :

1. Take over the configurations of the RootContainer
2. Default PortalContainer configurations from all JAR files (folder */conf/portal/configuration.xml*)
3. Web application configurations from the portal.war file - or the *portal* weppapp (folder */WEB-INF/conf/configuration.xml*)
4. External configuration for services of a named portal, it will be found at *exo-tomcat/exo-conf/portal/$portal_name/configuration.xml* (as of Portal 2.5)

You see, here the */conf/portal/configuration.xml* file of each jar enters the game, they are searched at first. Next, there is nearly always a configuration.xml in the portal.war file (or in the portal webapp folder), you find this file at */WEB-INF/conf/configuration.xml*. If you open it, you will find a lot of import statements that point to other configuration files in the same portal.war (or portal webapp).

Multiple Portals Be aware that you might set up several different portals ("admin", "mexico", etc.), and each of these portals will use a different PortalContainer. And each of these PortalContainers can be configured separately. As of eXo Portal 2.5 you also will be able to provide configurations from outside the jars and wars or webapps. Put a configuration file in *exo-tomcat/exo-conf/*

portal/$portal_name/configuration.xml where `$portal_name` is the name of the portal you want to configure for . But normally you only have one portal which is called "portal" so you use *exo-tomcat/exo-conf/portal/portal/configuration.xml*.

Note

As of eXo Portal 2.5 you can override the external configuration location with the system property *exo.conf.dir*. If the property exists its value will be used as path to the eXo configuration directory, that means this is an alternative to *exo-tomcat/exo-conf*. Just put this property in the command line: *java -Dexo.conf.dir=/path/to/exo/conf* or use eXo.bat or eXo.sh. In this particular use case, you have no need to use any prefixes in your configuration file to import other files. For example, if your configuration file is *exo-tomcat/exo-conf/portal/PORTAL_NAME/configuration.xml* and you want to import the configuration file *exo-tomcat/exo-conf/portal/PORTAL_NAME/mySubConfDir/myConfig.xml*, you can do it by adding *<import>mySubConfDir/myConfig.xml</import>* to your configuration file.

Note

Under **JBoss** application server *exo-conf* will be looked up in directory described by JBoss System property *jboss.server.config.url*. If the property is not found or empty *exo-jboss/exo-conf* will be asked (since kernel 2.0.4).

StandaloneContainer

In the same way as the PortalContainer the StandaloneContainer *takes over the configuration of the RootContainer*. After that our configuration gets a little bit more tricky because standalone containers can be initialized using an URL. This URL contains a link to an external configuration. As you probably never need a standalone configuration you can safely jump over the remaining confusing words of this chapter.

After taking over RootContainer's configuration, there are three cases which depend on the URL initialization, :

- **Independent configuration by URL** No other configuration file is taken in consideration. The configuration provided by the URL is used without any default configs. That means that the container creates a new empty hashtable and not any bit of previous configuration is used. Apply the following code to do this:

```
StandaloneContainer.setConfigurationURL(containerConf);
```

- **Additional configuration by URL** The StandaloneContainer is initialized very similar to the PortalContainer, but the last step is slightly different. A configuration file that is provided by the URL is used to replace some of the service configurations.The code looks like this:

```
StandaloneContainer.addConfigurationURL(containerConf);
```

1. Take over the configurations of the RootContainer
2. Default *StandaloneContainer* configurations from JAR files (folder */conf/portal/configuration.xml*)
3. Web application configurations from WAR files (folder */WEB-INF/conf/configuration.xml*)
4. Configuration from added URL *containerConf* overrides only services configured in the file

- **File based configuration** No URL is involved, in this case the sequence is:

1. Take over the configurations of the RootContainer
2. Default *StandaloneContainer* configurations from JAR files (folder */conf/portal/configuration.xml*)
3. Web applications configurations from WAR files (folder */WEB-INF/conf/configuration.xml*)
4. External configuration for *StandaloneContainer* services, it will be found at *$user_home/exo-configuration.xml*. If *$user_home/exo-configuration.xml* doesn't exist and the *StandaloneContainer* instance obtained with the dedicated configuration classloader the container will try to retrieve the resource *conf/exo-configuration.xml* within the given classloader (user_home is your home directory like "C:/Documents and Settings/Smith").

Service instantiation

As you have already learned the services are all singletons, so that the container creates only one single instance of each container. The services are created by calling the constructors (called *constructor injection*). If there are only zero-arguments constructors (`Foo public Foo(){}`) there are no problems to be expected. That's easy.

But now look at *https://anonsvn.jboss.org/repos/exo-jcr/core/trunk/exo.core.component.organization.jdbc/src/main/java/org/exoplatform/services/organization/jdbc/OrganizationServiceImpl.java*

This JDBC implementation of BaseOrganizationService interface has only one constructor:

```
public OrganizationServiceImpl(ListenerService listenerService, DatabaseService dbService);
```

You see this service depends on two other services. In order to be able to call this constructor the container first needs a `ListenerService` and a `DatabaseService`. Therefore these services must be instantiated before `BaseOrganizationService`, because `BaseOrganizationService` depends on them.

For this purpose the container first looks at the constructors of all services and creates a matrix of service dependencies in order to call the services in a proper order. If for any reason there are interdependencies or circular dependencies you will get a java `Exception`. *In this way the dependencies are injected by the container*.

Note

What happens if one service has more than one constructor? The container always tries first to use the constructor with a maximum of arguments, if this is not possible the container continues step by step with constructors that have less arguments until arriving at the zero-argument constructor (if there is one).

Miscellaneous

Startable interface

Your service can implement the *startable* interface which defines a *start()* and a *stop()* method. These methods are called by the container at the beginning and the end of the container's lifecycle. This way the lifecycle of your service is managed by the container.

Inversion of Control

Retrospection Do you remember your last project where you had some small components and several larger services? How was this organized? Some services had their own configuration files, others had static values in the source code. Most components were probably tightly coupled to the main application, or you called static methods whenever you needed a service in your java class. Presumably you even copied the source code of an earlier project in order to adapt the implementation to your needs. In short:

- Each of your service had a proprietary configuration mechanism.
- The service lifecycles were managed inside of each service or were arbitrary.
- The dependencies between your services were implementation-dependent and tightly coupled in your source code.

New Approach You have seen that eXo uses the *Inversion of Control* (IoC) pattern which means that the control of the services is given to an independent outside entity, in this case a *container*. Now the container takes care of everything:

- The *configuration is injected* by external configuration files.
- The *lifecycle is managed from outside*, because the constructors are called by the container. You can achieve an even finer lifecycle management if you use the startable interface.
- The *dependencies are injected* by the service instantiation process.

Dependency Injection You also saw two types of dependency injections:

- Constructor injection: The constructor is called by the container.
- Setter injection: Whenever you use *external-plugins* to provide your service with plugins (see *Service Configuration in Detail*.

More Containers

There are two more Containers called `RepositoryContainer` and `WorkspaceContainer`. These are specificities of eXo JCR, for the sake of simplicity. You don't need them.

Single Implementation Services

In some case the developer of a service does not expect that there will be several implementations for his service. Therefore he does not create an interface. In this case the configuration looks like this:

```
<key>org.exoplatform.services.database.jdbc.DBSchemaCreator</key>
<type>org.exoplatform.services.database.jdbc.DBSchemaCreator</type>
```

The key and type tags contain equally the qualified class name.

Configuration properties

Since kernel 2.0.7 and 2.1, it is possible to use system properties in literal values of component configuration meta data. Thus it is possible to resolve properties at runtime instead of providing a value at packaging time.

```
<component>
  ...
  <init-params>
    <value-param>
      <name>simple_param</name>
      <value>${simple_param_value}</value>
    </value-param>
    <properties-param>
      <name>properties_param</name>
      <property name="value_1" value="properties_param_value_1"/>
      <property name="value_2" value="${properties_param_value_2}"/>
    </properties-param>
    <object-param>
      <name>object_param</name>
      <object type="org.exoplatform.xml.test.Person">
        <field name="address"><string>${person_address}</string></field>
        <field name="male"><boolean>${person_male}</boolean></field>
```

```
      <field name="age"><int>${age_value}</int></field>
      <field name="size"><double>${size_value}</double></field>
    </object>
   </object-param>
  </init-params>
</component>
```

Configuration Logging

In case you need to solve problems with your service configuration, you have to know from which JAR/WAR causes your troubles. Add the JVM system property `org.exoplatform.container.configuration.debug` to your eXo.bat or eXo.sh file (exo-tomcat/bin/).

```
set EXO_CONFIG_OPTS="-Dorg.exoplatform.container.configuration.debug"
```

If this property is set the container configuration manager reports during startup the configuration retrieval process to the standard output (System.out).

```
.....
Add configuration jar:file:/D:/Projects/eXo/dev/exo-working/exo-tomcat/lib/exo.kernel.container-trunk.jar!/conf/portal/configuration.xml
Add configuration jar:file:/D:/Projects/eXo/dev/exo-working/exo-tomcat/lib/exo.kernel.component.cache-trunk.jar!/conf/portal/configuration.xml
Add configuration jndi:/localhost/portal/WEB-INF/conf/configuration.xml import jndi:/localhost/portal/WEB-INF/conf/common/common-configuration.xml
import jndi:/localhost/portal/WEB-INF/conf/database/database-configuration.xml import jndi:/localhost/portal/WEB-INF/conf/ecm/jcr-component-plugins-configuration.xml
import jndi:/localhost/portal/WEB-INF/conf/jcr/jcr-configuration.xml
.....
```

Further Reading

Do you feel an expert now? Not yet. Get a deeper look and read this *Services Wiring* article. You read so much about configuration, that you should wonder what the *XML Schema of the configuration file* looks like.

If you wish to see a examples of service configurations you should study the *Core.* Where you find descriptions of some eXo's core services. Finally you might wish to read more about *PicoContainer* [http://www.picocontainer.org/].

Service Configuration in Detail

Related documents

- *Service Configuration for Beginners*
- *Services Wiring*
- *Kernel Configuration File*

Objectives

This article shows how to setup a sample service with some configurations and how to access to the configuration parameters. The later chapters describe all details of the configuration file (parameters, object-params, plugins, imports, etc.), it also shows how to access the configuration values. You may consider this article as a **reference**, but you can also use this article as a **tutorial** and read it from the beginning to the end.

Requirements

You should have read and understood *Service Configuration for Beginners*. Obviously you should know java and xml. We are working with examples that are created for teaching reasons only and you will see extracts from the eXo Products default installation. When reading this article, you do not forget that the terms service and component are interchangeable in eXo Products.

Sample Service

Java Class

Imagine that you are working for a publishing company called "La Verdad" that is going to use eXo platform. Your boss asks you be able to calculate the number of sentences of an article.

You remember in eXo product everything is a **service** so you decide to create a simple class. In the future, you want to be able to plug different implementations of your service, so that you should define an **interface** that defines your service.

```
package com.laverdad.services;
public interface ArticleStatsService {
  public abstract int calcSentences(String article);
}
```

A very simple implementation:

```
public class ArticleStatsServiceImpl implements ArticleStatsService {
  public int calcSentences(String article) {
    throw new RuntimeException("Not implemented");
  }
}
```

That's it! You see there are no special prerequisites for a service.

You should already have prepared your working environment, where you have a base folder (let's call it our service base folder). If you wish to try out this example create this class in the com/laverdad/services/ArticleStatsService subfolder.

First configuration file

When creating a service, you also should declare its existence to the **Container**, therefore you create a first simple configuration file. Copy the following code to a file called "configuration.xml" and place this file in a /conf subdirectory of your service base folder. As you already know the container looks for a "/conf/configuration.xml" file in each jar-file.

```
<?xml version="1.0" encoding="UTF8"?>
<configuration
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd      http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
   xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
  <component>
    <key>com.laverdad.services.ArticleStatsService</key>
    <type>com.laverdad.services.ArticleStatsServiceImpl</type>
  </component>
</configuration>
```

Note

You are correctly using the namespace of the configuration schema (`http://www.exoplatform.org/xml/ns/kernel_1_2.xsd`). Most of the configuration schema is explained in this article, therefore you do not need to open and understand the schema. For backward compatibility it is not necessary to declare the schema.

When eXo kernel reads a configuration, it loads the file from the kernel jar using the classloader and does not use an internet connection to resolve the file.

Init Parameters

You see your service has a configuration file, but you wonder how the file could possibly access to its configuration. Imagine that you are asked to implement two different calculation methods: fast and exact.

You create one init parameter containing the calculation methods. For the exact method, you wish to configure more details for the service. Let's enhance the word service configuration file:

```
<component>
  <key>com.laverdad.services.ArticleStatsService</key>
  <type>com.laverdad.services.ArticleStatsServiceImpl</type>
  <init-params>
    <value-param>
      <name>calc-method</name>
      <description>calculation method: fast, exact</description>
      <value>fast</value>
    </value-param>
    <properties-param>
      <name>details-for-exact-method</name>
      <description>details for exact phrase counting</description>
      <property name="language" value="English" />
      <property name="variant" value="us" />
    </properties-param>
  </init-params>
</component>
```

Note

When configuring your service, you are **totally free**. You can provide as many **value-param**, **property-param**, and **properties** you wish and you can give them any names or values. You only must respect the xml structure.

Now let's see how our service can read this configuration. The implementation of the calcSentences() method serves just as a simple example. It's up to your imagination to implement the exact method.

```
public class ArticleStatsServiceImpl implements ArticleStatsService {

  private String calcMethod = "fast";
  private String variant = "French";
  private String language = "France";
```

```
  public ArticleStatsServiceImpl(InitParams initParams) {
    super();
    calcMethod = initParams.getValueParam("calc-method").getValue();
     PropertiesParam detailsForExactMethod = initParams.getPropertiesParam("details-for-exact-
method");
    if ( detailsForExactMethod != null) {
      language = detailsForExactMethod.getProperty("language");
      variant = detailsForExactMethod.getProperty("variant");
    }
  }

  public int calcSentences(String article) {
    if (calcMethod == "fast") {
      // just count the number of periods "."
      int res = 0;
      int period = article.indexOf('.');
      while (period != -1) {
        res++;
        article = article.substring(period+1);
        period = article.indexOf('.');
      }
      return  res;
    }
    throw new RuntimeException("Not implemented");
    }
}
```

You see you just have to declare a parameter of org.exoplatform.container.xml.InitParams in your constructor. The container provides an InitParams object that correspond to the xml tree of init-param.

Service Access

As you want to follow the principle of **Inversion of Control,** you **must not** access the service directly. You need a **Container** to access the service.

With this command you get your current container:

- **ExoContainer myContainer = ExoContainerContext.getCurrentContainer();**

This might be a PortalContainer or a StandaloneContainer, dependant on the *execution mode* in which you are running your application.

Whenever you need one of the services that you have configured use the method:

- **myContainer.getComponentInstance(class)**

In our case:

- **ArticleStatsService statsService = (ArticleStatsService) myContainer.getComponentInstance(ArticleStatsService.class);**

Recapitulation:

```
package com.laverdad.common;

import org.exoplatform.container.ExoContainer;
import org.exoplatform.container.ExoContainerContext;
import com.laverdad.services.*;

public class Statistics {

  public int makeStatistics(String articleText) {
    ExoContainer myContainer = ExoContainerContext.getCurrentContainer();
    ArticleStatsService statsService = (ArticleStatsService)
        myContainer.getComponentInstance(ArticleStatsService.class);
    int numberOfSentences = statsService.calcSentences(articleText);
    return numberOfSentences;
  }

  public static void main( String args[]) {
   Statistics stats = new Statistics();
   String newText = "This is a normal text. The method only counts the number of periods. "
   + "You can implement your own implementation with a more exact counting. "
   + "Let`s make a last sentence.";
  System.out.println("Number of sentences: " + stats.makeStatistics(newText));
  }
}
```

If you test this sample in standalone mode, you need to put all jars of eXo Kernel in your buildpath, furthermore picoContainer is needed.

Parameters

Value-Param

There is an value-param example:

```
 <component>
  <key>org.exoplatform.portal.config.UserACL</key>
```

```
  <type>org.exoplatform.portal.config.UserACL</type>
  <init-params>
...
    <value-param>
      <name>access.control.workspace</name>
        <description>groups with memberships that have the right to access the User Control
 Workspace</description>
      <value>*:/platform/administrators,*:/organization/management/executive-board</value>
    </value-param>
...
 </component>
```

The UserACL class accesses to the **value-param** in its constructor.

```
package org.exoplatform.portal.config;
public class UserACL {

 public UserACL(InitParams params) {
  UserACLMetaData md = new UserACLMetaData();
                                    ValueParam accessControlWorkspaceParam =
 params.getValueParam("access.control.workspace");
                                      if(accessControlWorkspaceParam != null)
 md.setAccessControlWorkspace(accessControlWorkspaceParam.getValue());
...
```

Properties-Param

Properties are name-value pairs. Both the name and the value are Java Strings.

Here you see the hibernate configuration example:

```
 <component>
  <key>org.exoplatform.services.database.HibernateService</key>
  <type>org.exoplatform.services.database.impl.HibernateServiceImpl</type>
  <init-params>
   <properties-param>
    <name>hibernate.properties</name>
    <description>Default Hibernate Service</description>
    <property name="hibernate.show_sql" value="false"/>
    <property name="hibernate.cglib.use_reflection_optimizer" value="true"/>
    <property name="hibernate.connection.url" value="jdbc:hsqldb:file:../temp/data/exodb"/>
    <property name="hibernate.connection.driver_class" value="org.hsqldb.jdbcDriver"/>
```

```
...
      </properties-param>
    </init-params>
  </component>
```

In the org.exoplatform.services.database.impl.HibernateServiceImpl you will find that the name "hibernate.properties" of the properties-param is used to access the properties.

```
package org.exoplatform.services.database.impl;

public class HibernateServiceImpl implements HibernateService, ComponentRequestLifecycle {
  public HibernateServiceImpl(InitParams initParams, CacheService cacheService) {
    PropertiesParam param = initParams.getPropertiesParam("hibernate.properties");
...
}
```

Object-Param

Let's have a look at the configuration of the LDAPService. It's not important to know LDAP, we only discuss the parameters.

```
<component>
    <key>org.exoplatform.services.ldap.LDAPService</key>
    <type>org.exoplatform.services.ldap.impl.LDAPServiceImpl</type>
    <init-params>
      <object-param>
        <name>ldap.config</name>
        <description>Default ldap config</description>
        <object type="org.exoplatform.services.ldap.impl.LDAPConnectionConfig">
   <field  name="providerURL"><string>ldaps://10.0.0.3:636</string></field>
         <field      name="rootdn"><string>CN=Administrator,CN=Users,DC=exoplatform,DC=org</
string></field>
   <field  name="password"><string>exo</string></field>
   <field  name="version"><string>3</string></field>
     <field  name="minConnection"><int>5</int></field>
       <field  name="maxConnection"><int>10</int></field>
       <field  name="referralMode"><string>ignore</string></field>
       <field  name="serverName"><string>active.directory</string></field>
       </object>
      </object-param>
    </init-params>
```

```
</component>
```

You see here an **object-param** is being used to pass the parameters inside an object (actually a java bean). It consists of a **name**, a **description** and exactly one **object**. The object defines the **type** and a number of **fields**.

Here you see how the service accesses the object:

```
package org.exoplatform.services.ldap.impl;

public class LDAPServiceImpl implements LDAPService {
...
  public LDAPServiceImpl(InitParams params) {
                              LDAPConnectionConfig config = (LDAPConnectionConfig)
 params.getObjectParam("ldap.config")
                                                  .getObject();
...
```

The passed object is LDAPConnectionConfig which is a classic **java bean**. It contains all fields and also the appropriate getters and setters (not listed here). You also can provide default values. The container creates a new instance of your bean and calls all setters whose values are configured in the configuration file.

```
package org.exoplatform.services.ldap.impl;

public class LDAPConnectionConfig {
  private String providerURL       = "ldap://127.0.0.1:389";
  private String rootdn;
  private String password;
  private String version;
  private String authenticationType = "simple";
  private String serverName         = "default";
  private int    minConnection;
  private int    maxConnection;
  private String referralMode       = "follow";
...
```

You see that the types (String, int) of the fields in the configuration correspond with the bean. A short glance in the kernel_1_0.xsd file let us discover more simple types:

- **string, int, long, boolean, date, double**

Have a look on this type test xml file: *https://anonsvn.jboss.org/repos/exo-jcr/kernel/trunk/exo.kernel.container/src/test/resources/object.xml*.

Collection

You also can use java collections to configure your service. In order to see an example, let's open the database-organization-configuration.xml file. This file defines a default user organization (users, groups, memberships/roles) of your portal. They use component-plugins which are explained later. You wil see that object-param is used again.

There are two collections: The first collection is an **ArrayList**. This ArrayList contains only one value, but there could be more. The only value is an object which defines the field of the NewUserConfig$JoinGroup bean.

The second collection is a **HashSet** that is a set of strings.

```
<component-plugin>
  <name>new.user.event.listener</name>
  <set-method>addListenerPlugin</set-method>
  <type>org.exoplatform.services.organization.impl.NewUserEventListener</type>
  <description>this listener assign group and membership to a new created user</description>
  <init-params>
    <object-param>
      <name>configuration</name>
      <description>description</description>
      <object type="org.exoplatform.services.organization.impl.NewUserConfig">
        <field  name="group">
          <collection type="java.util.ArrayList">
            <value>
           <object type="org.exoplatform.services.organization.impl.NewUserConfig$JoinGroup">
                <field  name="groupId"><string>/platform/users</string></field>
                <field  name="membership"><string>member</string></field>
              </object>
            </value>
          </collection>
        </field>
        <field  name="ignoredUser">
          <collection type="java.util.HashSet">
            <value><string>root</string></value>
            <value><string>john</string></value>
            <value><string>marry</string></value>
            <value><string>demo</string></value>
            <value><string>james</string></value>
          </collection>
        </field>
```

```
        </object>
      </object-param>
    </init-params>
  </component-plugin>
```

Let's look at the org.exoplatform.services.organization.impl.NewUserConfig bean:

```
public class NewUserConfig {
  private List    role;
  private List    group;
  private HashSet ignoredUser;

  ...

  public void setIgnoredUser(String user) {
    ignoredUser.add(user);

  ...

  static public class JoinGroup {
    public String  groupId;
    public String  membership;
  ...
}
```

You see the values of the HashSet are set one by one by the container, and it's the responsibility of the bean to add these values to its HashSet.

The JoinGroup object is just an inner class and implements a bean of its own. It can be accessed like any other inner class using NewUserConfig.JoinGroup.

External Plugin

The External Plugin allows you to add configuration on the fly.

As you have carefully read *Service Configuration for Beginners* you know that **normally** newer configurations always **replaces** previous configurations. An external plugin allows you to **add** configuration without replacing previous configurations.

That can be interesting if you adapt a service configuration for your project-specific needs (country, language, branch, project, etc.).

Let's have a look at the configuration of the TaxonomyPlugin of the CategoriesService:

```
<external-component-plugins>
          <target-component>org.exoplatform.services.cms.categories.CategoriesService</target-component>
   <component-plugin>
    <name>predefinedTaxonomyPlugin</name>
     <set-method>addTaxonomyPlugin</set-method>
     <type>org.exoplatform.services.cms.categories.impl.TaxonomyPlugin</type>
     <init-params>
      <value-param>
        <name>autoCreateInNewRepository</name>
        <value>true</value>
       </value-param>
       <value-param>
        <name>repository</name>
        <value>repository</value>
       </value-param>
      <object-param>
       <name>taxonomy.configuration</name>
         <description>configuration predefined taxonomies to inject in jcr</description>
         <object type="org.exoplatform.services.cms.categories.impl.TaxonomyConfig">
          <field  name="taxonomies">
           <collection type="java.util.ArrayList">
             <!-- cms taxonomy -->
            <value>
                                                                                <object type="org.exoplatform.services.cms.categories.impl.TaxonomyConfig$Taxonomy">
               <field  name="name"><string>cmsTaxonomy</string></field>
              <field  name="path"><string>/cms</string></field>
             </object>
            </value>
            <value>
                                                                                <object type="org.exoplatform.services.cms.categories.impl.TaxonomyConfig$Taxonomy">
               <field  name="name"><string>newsTaxonomy</string></field>
              <field  name="path"><string>/cms/news</string></field>
             </object>
            </value>
          </field>
         </object>
      </object-param>
    </init-params>
  </component-plugin>
<external-component-plugins>
```

The **<target-component>** defines the service for which the plugin is defined. The configuration is injected by the container using a method that is defined in **<set-method>**. The method has exactly one argument of the type org.exoplatform.services.cms.categories.impl.TaxonomyPlugin:

- addTaxonomyPlugin(org.exoplatform.services.cms.categories.impl.TaxonomyPlugin plugin)

The content of **<init-params>** corresponds to the structure of the TaxonomyPlugin object.

Note

You can configure the component CategoriesService using the addTaxonomyPlugin as often as you wish, you can also call addTaxonomyPlugin in different configuration files. The method addTaxonomyPlugin is then called several times, everything else depends on the implementation of the method.

Import

The import tag allows to link to other configuration files. These imported files can be placed anywhere. If you write a default configuration which is part of your jar file you should not import files from outside your jar.

- **war**: Imports from **portal.war/WEB-INF**
- **jar** or **classpath**: Uses the **classloader**, you can use this prefix in the default configuration for importing an other configuration file which is accessible by the classloader.
- **file**: Uses an **absolute path**, you also can put a **URL**.
- **without any prefix**:
  - Standalone mode: **user directory**
  - Portal mode: $AS-HOME, that means the application server home, for example " **exo-tomcat**".

If you open the "portal/trunk/web/portal/src/main/webapp/WEB-INF/conf.configuration.xml" you will see that it consists only of imports:

```
<import>war:/conf/common/common-configuration.xml</import>
<import>war:/conf/common/logs-configuration.xml</import>
<import>war:/conf/database/database-configuration.xml</import>
<import>war:/conf/jcr/jcr-configuration.xml</import>
<import>war:/conf/common/portlet-container-configuration.xml</import>
...
```

System properties

Since kernel 2.0.7 and 2.1, it is possible to use system properties in literal values of component configuration meta data. This makes it possible to resolve properties at runtime instead of providing a value at packaging time.

In portal/trunk/web/portal/src/main/webapp/WEB-INF/conf/database/database-configuration.tmpl.xml you find an example for system properties:

```
 <component>
   <key>org.exoplatform.services.database.HibernateService</key>
   <jmx-name>database:type=HibernateService</jmx-name>
   <type>org.exoplatform.services.database.impl.HibernateServiceImpl</type>
   <init-params>
     <properties-param>
       <name>hibernate.properties</name>
       <description>Default Hibernate Service</description>
...
       <property name="hibernate.connection.url" value="${connectionUrl}"/>
       <property name="hibernate.connection.driver_class" value="${driverClass}"/>
       <property name="hibernate.connection.username" value="${username}"/>
       <property name="hibernate.connection.password" value="${password}"/>
       <property name="hibernate.dialect" value="${dialect}"/>
...
     </properties-param>
   </init-params>
 </component>
```

As these are system properties you use the -D command: **java -DconnectionUrl=jdbc:hsqldb:file:../temp/data/exodb -DdriverClass=org.hsqldb.jdbcDriver** Or better use the parameters of eXo.bat / eXo.sh when you start eXo Portal: **set EXO_OPTS="-DconnectionUrl=jdbc:hsqldb:file:../temp/data/exodb -DdriverClass=org.hsqldb.jdbcDriver"**

Container Configuration

Intro

eXo Portal uses PicoContainer, which implements the Inversion of Control (IoC) design pattern. All eXo containers inherit from a PicoContainer. There are mainly two eXo containers used, each of them can provide one or several services. Each container service is delivered in a JAR file. This JAR file may contain a default configuration. The use of default configurations is recommended and most services provide it.

When a Pico Container searches for services and its configurations, each configurable service may be reconfigured to override default values or set additional parameters. If the service is configured in two or more places the configuration override mechanism will be used.

Confused? - You might be interested in the *Service Configuration for Beginners* article, which explains the basics.

Kernel configuration namespace

To be effective, the namespace URI `http://www.exoplatform.org/xml/ns/kernel_1_2.xsd` must be target namespace of the XML configuration file.

```
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd  http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
          xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">


  ...
</configuration>
```

Note

Any values in the configuration files can be created thanks to variables since the eXo kernel resolves them, for example the following configuration will be well interpreted:

```
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd    http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
```

```
  <import>${db.configuration.path}/db.xml</import>
  <import>${java.io.tmpdir}/bindfile.xml</import>
  <import>simple.xml</import>

</configuration>
```

The variables that are supported, are System properties and variables that are specific to your portal container, see next chapters for more details.

Understanding how configuration files are loaded

eXo Portal uses PicoContainer, which implements the Inversion of Control (IoC) design pattern. All eXo containers inherit from a PicoContainer. There are mainly two eXo containers used, each of them can provide one or several services. Each container service is delivered in a JAR file. This JAR file may contain a default configuration. The use of default configurations is recommended and most of services provide it.

When a Pico Container searches for services and its configurations, each configurable service may be reconfigured to override default values or set additional parameters. If the service is configured in two or more places, the configuration override mechanism will be used.

Configuration Retrieval

The container performs the following steps to make eXo Container configuration retrieval, depending on the container type.

Configuration retrieval order for the `PortalContainer`

The container is initialized by looking into different locations. This container is used by portal applications. Configurations are overloaded in the following lookup sequence:

1. Services default `RootContainer` configurations from JAR files */conf/configuration.xml*
2. External `RootContainer` configuration can be found at *$AS_HOME/exo-conf/configuration.xml*
3. Services default `PortalContainer` configurations from JAR files */conf/portal/configuration.xml*
4. Web applications configurations from WAR files */WEB-INF/conf/configuration.xml*
5. External configuration for services of named portal can be found at *$AS_HOME/exo-conf/portal/$PORTAL_NAME/configuration.xml*

Configuration retrieval for a `StandaloneContainer`

The container is initialized by looking into different locations. This container is used by non portal applications. Configurations are overloaded in the following lookup sequence:

1. Services default `RootContainer` configurations from JAR files */conf/configuration.xml*
2. External `RootContainer` configuration can be found at *$AS_HOME/exo-conf/configuration.xml*
3. Services default `StandaloneContainer` configurations from JAR files */conf/portal/configuration.xml*
4. Web applications configurations from WAR files */WEB-INF/conf/configuration.xml*
5. Then depending on the `StandaloneContainer` configuration URL initialization:
   - if configuration URL was initialized to be added to services defaults, as below:

```
// add configuration to the default services configurations from JARs/WARs
StandaloneContainer.addConfigurationURL(containerConf);
```

     Configuration from added URL *containerConf* will override only services configured in the file

   - if configuration URL not initialized at all, it will be found at *$AS_HOME/exo-configuration.xml*. If *$AS_HOME/exo-configuration.xml* doesn't exist the container will try find it at *$AS_HOME/exo-conf/exo-configuration.xml* location and if it's still not found and the `StandaloneContainer` instance obtained with the dedicated configuration `ClassLoader` the container will try to retrieve the resource *conf/exo-configuration.xml* within the given `ClassLoader`.

General notes about the configuration retrieval

Note

$AS_HOME - application server home directory, or *user.dir* JVM system property value in case of Java Standalone application. The application server home is:

- For `Jonas`, the value of the variable *${jonas.base}.*
- For `Jetty`, the value of the variable *${jetty.home}.*
- For `Websphere`, the value of the variable *${was.install.root}.*
- For `Weblogic`, the value of the variable *${wls.home}*.
- For `Glassfish`, the value of the variable *${com.sun.aas.instanceRoot}*.
- For `Tomcat`, the value of the variable *${catalina.home}*.
- For `JBoss AS`, the value of the variable *${jboss.server.config.url}* if the exo-conf directory can be found there otherwise it will be the value of the variable *${jboss.home.dir}*.

Note

$PORTAL_NAME - portal web application name.

Note

External configuration location can be overridden with System property *exo.conf.dir*. If the property exists, its value will be used as path to eXo configuration directory, i.e. to *$AS_HOME/exo-conf* alternative. E.g. put property in command line java -*Dexo.conf.dir=/path/to/exo/conf*. In this particular use case, you do not need to use any prefix to import other files. For instance, if your configuration file is *$AS_HOME/exo-conf/portal/PORTAL_NAME/configuration.xml* and you want to import the configuration file *$AS_HOME/exo-conf/portal/PORTAL_NAME/mySubConfDir/myConfig.xml*, you can do it by adding *<import>mySubConfDir/myConfig.xml</import>* to your configuration file.

Note

The name of the configuration folder that is by default *"exo-conf"*, can be changed thanks to the System property *exo.conf.dir.name*.

Note

The search looks for a configuration file in each JAR/WAR available from the classpath using the current thread context classloader. During the search these configurations are added to a set. If the service was configured previously and the current JAR contains a new configuration of that service the latest (from the current JAR/WAR) will replace the previous one. The last one will be applied to the service during the services start phase.

Take care to have no dependencies between configurations from JAR files (*/conf/portal/configuration.xml* and */conf/configuration.xml*) since we have no way to know in advance the loading order of those configurations. In other words, if you want to overload some configuration located in the file */conf/portal/configuration.xml* of a given JAR file, you must not do it from the file */conf/portal/configuration.xml* of another JAR file but from another configuration file loaded after configurations from JAR files */conf/portal/configuration.xml.*

After the processing of all configurations available in system, the container will initialize it and start each service in order of the dependency injection (DI).

The user/developer should be careful when configuring the same service in different configuration files. It's recommended to configure a service in its own JAR only. Or, in case of a portal configuration, strictly reconfigure the services in portal WAR files or in an external configuration.

There are services that can be (or should be) configured more than one time. This depends on business logic of the service. A service may initialize the same resource (shared with other services) or may add a particular object to a set of objects (shared with other services too). In the

first case, it's critical who will be the last, i.e. whose configuration will be used. In the second case, it's no matter who is the first and who is the last (if the parameter objects are independent).

Configuration retrieval log

In case of problems with service configuration, it's important to know from which JAR/WAR it comes. For that purpose, the JVM system property *org.exoplatform.container.configuration.debug* can be used.

```
java -Dorg.exoplatform.container.configuration.debug ...
```

If the property is enabled, the container configuration manager will log the configuration adding process at *INFO* level.

```
......
  Add configuration jar:file:/D:/Projects/eXo/dev/exo-working/exo-tomcat/lib/exo.kernel.container-trunk.jar!/conf/portal/configuration.xml
                Add configuration jar:file:/D:/Projects/eXo/dev/exo-working/exo-tomcat/lib/exo.kernel.component.cache-trunk.jar!/conf/portal/configuration.xml
  Add configuration jndi:/localhost/portal/WEB-INF/conf/configuration.xml
      import jndi:/localhost/portal/WEB-INF/conf/common/common-configuration.xml
      import jndi:/localhost/portal/WEB-INF/conf/database/database-configuration.xml
      import jndi:/localhost/portal/WEB-INF/conf/ecm/jcr-component-plugins-configuration.xml
      import jndi:/localhost/portal/WEB-INF/conf/jcr/jcr-configuration.xml
  ......
```

Getting the effective configuration at Runtime

The effective configuration of the StandaloneContainer, RootContainer and/or PortalContainer can be known thanks to the method *getConfigurationXML*() that is exposed through JMX at the container's level. This method will give you the effective configuration in XML format that has been really interpreted by the kernel. This could be helpful to understand how a given component or plugin has been initialized.

Advanced concepts for the *PortalContainers*

Since eXo JCR 1.12, we added a set of new features that have been designed to extend portal applications such as GateIn.

Add new configuration files from a WAR file

A `ServletContextListener` called `org.exoplatform.container.web.PortalContainerConfigOwner` has been added in order to notify the application that a given web application provides some configuration to the portal

container, and this configuration file is the file *WEB-INF/conf/configuration.xml* available in the web application itself.

If your war file contains some configuration to add to the `PortalContainer` simply add the following lines in your *web.xml* file.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
                "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
...
  <!-- ======================================================================= -->
  <!--          LISTENER                                                       -->
  <!-- ======================================================================= -->
  <listener>
    <listener-class>org.exoplatform.container.web.PortalContainerConfigOwner</listener-class>
  </listener>
...
</web-app>
```

Creating your *PortalContainers* from a WAR file

A `ServletContextListener` called `org.exoplatform.container.web.PortalContainerCreator` has been added in order to create the current portal containers that have been registered. We assume that all the web applications have already been loaded before calling `PortalContainerCreator.contextInitialized[.]`

Note

In GateIn, the `PortalContainerCreator` is already managed by the file *starter.war/ear.*

Defining a *PortalContainer* with its dependencies and its settings

Now we can define precisely a portal container and its dependencies and settings thanks to the `PortalContainerDefinition` that currently contains the name of the portal container, the name of the rest context, the name of the realm, the web application dependencies ordered by loading priority (i.e. the first dependency must be loaded at first and so on..) and the settings.

To be able to define a `PortalContainerDefinition`, we need to ensure first of all that a `PortalContainerConfig` has been defined at the `RootContainer` level, see an example below:

```
<component>
  <!-- The full qualified name of the PortalContainerConfig -->
  <type>org.exoplatform.container.definition.PortalContainerConfig</type>
```

```
<init-params>
  <!-- The name of the default portal container -->
  <value-param>
    <name>default.portal.container</name>
    <value>myPortal</value>
  </value-param>
  <!-- The name of the default rest ServletContext -->
  <value-param>
    <name>default.rest.context</name>
    <value>myRest</value>
  </value-param>
  <!-- The name of the default realm -->
  <value-param>
    <name>default.realm.name</name>
    <value>my-exo-domain</value>
  </value-param>
 <!-- Indicates whether the unregistered webapps have to be ignored -->
 <value-param>
    <name>ignore.unregistered.webapp</name>
    <value>true</value>
 </value-param>
  <!-- The default portal container definition -->
  <!-- It cans be used to avoid duplicating configuration -->
  <object-param>
    <name>default.portal.definition</name>
    <object type="org.exoplatform.container.definition.PortalContainerDefinition">
      <!-- All the dependencies of the portal container ordered by loading priority -->
      <field name="dependencies">
        <collection type="java.util.ArrayList">
          <value>
            <string>foo</string>
          </value>
          <value>
            <string>foo2</string>
          </value>
          <value>
            <string>foo3</string>
          </value>
        </collection>
      </field>
      <!-- A map of settings tied to the default portal container -->
      <field name="settings">
        <map type="java.util.HashMap">
          <entry>
```

```
            <key>
              <string>foo5</string>
            </key>
            <value>
              <string>value</string>
            </value>
          </entry>
          <entry>
            <key>
              <string>string</string>
            </key>
            <value>
              <string>value0</string>
            </value>
          </entry>
          <entry>
            <key>
              <string>int</string>
            </key>
            <value>
              <int>100</int>
            </value>
          </entry>
        </map>
      </field>
      <!-- The path to the external properties file -->
      <field name="externalSettingsPath">
       <string>classpath:/org/exoplatform/container/definition/default-settings.properties</string>
      </field>
    </object>
  </object-param>
 </init-params>
</component>
```

Table 53.1. Descriptions of the fields of `PortalContainerConfig`

default.portal.container (*)	The name of the default portal container. This field is optional.
default.rest.context (*)	The name of the default rest `ServletContext`. This field is optional.
default.realm.name (*)	The name of the default realm. This field is optional.
ignore.unregistered.webapp (*)	

	Indicates whether the unregistered webapps have to be ignored. If a webapp has not been registered as a dependency of any portal container, the application will use the value of this parameter to know what to do: • If it is set to *false*, this webapp will be considered by default as a dependency of all the portal containers. • If it is set to *true*, this webapp won't be considered by default as a dependency of any portal container, it will be simply ignored. This field is optional and by default this parameter is set to *false*.
default.portal.definition	The definition of the default portal container. This field is optional. The expected type is `org.exoplatform.container.definition.PortalContainerDef` that is described below. Allow the parameters defined in this default `PortalContainerDefinition` will be the default values.

Note

All the value of the parameters marked with a (*) can be defined thanks to System properties like any values in configuration files but also thanks to variables loaded by the *PropertyConfigurator*. For example in GateIn by default, it would be all the variables defined in the file *configuration.properties*.

A new `PortalContainerDefinition` can be defined at the `RootContainer` level thanks to an external plugin, see an example below:

```
 <external-component-plugins>
  <!-- The full qualified name of the PortalContainerConfig -->
        <target-component>org.exoplatform.container.definition.PortalContainerConfig</target-component>
  <component-plugin>
   <!-- The name of the plugin -->
   <name>Add PortalContainer Definitions</name>
    <!-- The name of the method to call on the PortalContainerConfig in order to register the PortalContainerDefinitions -->
   <set-method>registerPlugin</set-method>
```

```
<!-- The full qualified name of the PortalContainerDefinitionPlugin -->
<type>org.exoplatform.container.definition.PortalContainerDefinitionPlugin</type>
<init-params>
  <object-param>
    <name>portal</name>
    <object type="org.exoplatform.container.definition.PortalContainerDefinition">
      <!-- The name of the portal container -->
      <field name="name">
        <string>myPortal</string>
      </field>
      <!-- The name of the context name of the rest web application -->
      <field name="restContextName">
        <string>myRest</string>
      </field>
      <!-- The name of the realm -->
      <field name="realmName">
        <string>my-domain</string>
      </field>
      <!-- All the dependencies of the portal container ordered by loading priority -->
      <field name="dependencies">
        <collection type="java.util.ArrayList">
          <value>
            <string>foo</string>
          </value>
          <value>
            <string>foo2</string>
          </value>
          <value>
            <string>foo3</string>
          </value>
        </collection>
      </field>
      <!-- A map of settings tied to the portal container -->
      <field name="settings">
        <map type="java.util.HashMap">
          <entry>
            <key>
              <string>foo</string>
            </key>
            <value>
              <string>value</string>
            </value>
          </entry>
          <entry>
```

```
            <key>
              <string>int</string>
            </key>
            <value>
              <int>10</int>
            </value>
          </entry>
          <entry>
            <key>
              <string>long</string>
            </key>
            <value>
              <long>10</long>
            </value>
          </entry>
          <entry>
            <key>
              <string>double</string>
            </key>
            <value>
              <double>10</double>
            </value>
          </entry>
          <entry>
            <key>
              <string>boolean</string>
            </key>
            <value>
              <boolean>true</boolean>
            </value>
          </entry>
        </map>
      </field>
      <!-- The path to the external properties file -->
      <field name="externalSettingsPath">
        <string>classpath:/org/exoplatform/container/definition/settings.properties</string>
      </field>
    </object>
  </object-param>
 </init-params>
</component-plugin>
</external-component-plugins>
```

Table 53.2. Descriptions of the fields of a `PortalContainerDefinition` when it is used to define a new portal container

name (*)	The name of the portal container. This field is mandatory .
restContextName (*)	The name of the context name of the rest web application. This field is optional. The default value will be defined at the `PortalContainerConfig` level.
realmName (*)	The name of the realm. This field is optional. The default value will be defined at the `PortalContainerConfig` level.
dependencies	All the dependencies of the portal container ordered by loading priority. This field is optional. The default value will be defined at the `PortalContainerConfig` level. The dependencies are in fact the list of the context names of the web applications from which the portal container depends. This field is optional. The dependency order is really crucial since it will be interpreted the same way by several components of the platform. All those components, will consider the 1st element in the list less important than the second element and so on. It is currently used to: • Know the loading order of all the dependencies. • If we have several `PortalContainerConfigOwner` • The `ServletContext` of all the `PortalContainerConfigOwner` will be unified, if we use the unified `ServletContext` (*PortalContainer.getPortalContext()*) to get a resource, it will try to get the resource in the `ServletContext` of the most important `PortalContainerConfigOwner` (i.e. last in the dependency list) and if it cans find it, it will try with the second most important `PortalContainerConfigOwner` and so on.

	• The `ClassLoader` of all the `PortalContainerConfigOwner` will be unified, if we use the unified `ClassLoader` (*PortalContainer.getPortalClassLoader()*) to get a resource, it will try to get the resource in the `ClassLoader` of the most important `PortalContainerConfigOwner` (i.e. last in the dependency list) and if it can find it, it will try with the second most important `PortalContainerConfigOwner` and so on.
settings	A `java.util.Map` of internal parameters that we would like to tie the portal container. Those parameters could have any type of value. This field is optional. If some internal settings are defined at the `PortalContainerConfig` level, the two maps of settings will be merged. If a setting with the same name is defined in both maps, it will keep the value defined at the `PortalContainerDefinition` level.
externalSettingsPath	The path of the external properties file to load as default settings to the portal container. This field is optional. If some external settings are defined at the `PortalContainerConfig` level, the two maps of settings will be merged. If a setting with the same name is defined in both maps, it will keep the value defined at the `PortalContainerDefinition` level. The external properties files can be either of type "properties" or of type "xml". The path will be interpreted as follows: 1. The path doesn't contain any prefix of type "classpath:", "jar:" or "file:", we assume that the file could be externalized so we apply the following rules: a. A file exists at *${exo-conf-dir}/portal/${portalContainerName}/${externalSettingsPath}*, we will load this file.

	b. No file exists at the previous path, we then assume that the path cans be interpreted by the `ConfigurationManager`. 2. The path contains a prefix, we then assume that the path cans be interpreted by the `ConfigurationManager`.

Table 53.3. Descriptions of the fields of a `PortalContainerDefinition` when it is used to define the default portal container

name (*)	The name of the portal container. This field is optional. The default portal name will be: 1. If this field is not empty, then the default value will be the value of this field. 2. If this field is empty and the value of the parameter *default.portal.container* is not empty, then the default value will be the value of the parameter. 3. If this field and the parameter *default.portal.container* are both empty, the default value will be *"portal".*
restContextName (*)	The name of the context name of the rest web application. This field is optional. The default value wil be: 1. If this field is not empty, then the default value will be the value of this field. 2. If this field is empty and the value of the parameter *default.rest.context* is not empty, then the default value will be the value of the parameter. 3. If this field and the parameter *default.rest.context* are both empty, the default value will be *"rest".*
realmName (*)	The name of the realm. This field is optional. The default value wil be:

	1. If this field is not empty, then the default value will be the value of this field. 2. If this field is empty and the value of the parameter *default.realm.name* is not empty, then the default value will be the value of the parameter. 3. If this field and the parameter *default.realm.name* are both empty, the default value will be *"exo-domain".*
dependencies	All the dependencies of the portal container ordered by loading priority. This field is optional. If this field has a non empty value, it will be the default list of dependencies.
settings	A `java.util.Map` of internal parameters that we would like to tie the default portal container. Those parameters could have any type of value. This field is optional.
externalSettingsPath	The path of the external properties file to load as default settings to the default portal container. This field is optional. The external properties files can be either of type "properties" or of type "xml". The path will be interpreted as follows: 1. The path doesn't contain any prefix of type "classpath:", "jar:" or "file:", we assume that the file could be externalized so we apply the following rules: a. A file exists at *${exo-conf-dir}/portal/${externalSettingsPath}*, we will load this file. b. No file exists at the previous path, we then assume that the path cans be interpreted by the `ConfigurationManager`. 2. The path contains a prefix, we then assume that the path cans be interpreted by the `ConfigurationManager`.

Note

All the value of the parameters marked with a (*) can be defined thanks to System properties like any values in configuration files but also thanks to variables loaded by the *PropertyConfigurator*. For example in GateIn by default, it would be all the variables defined in the file *configuration.properties*.

Internal and external settings are both optional, but if we give a non empty value for both the application will merge the settings. If the same setting name exists in both settings, we apply the following rules:

1. The value of the external setting is *null*, we ignore the value.
2. The value of the external setting is not *null* and the value of the internal setting is *null*, the final value will be the external setting value that is of type `String`.
3. Both values are not `null`, we will have to convert the external setting value into the target type which is the type of the internal setting value, thanks to the static method *valueOf(String)*, the following sub-rules are then applied:
   a. The method cannot be found, the final value will be the external setting value that is of type `String`.
   b. The method can be found and the external setting value is an empty `String`, we ignore the external setting value.
   c. The method can be found and the external setting value is not an empty `String` but the method call fails, we ignore the external setting value.
   d. The method can be found and the external setting value is not an empty `String` and the method call succeeds, the final value will be the external setting value that is of type of the internal setting value.

`PortalContainer` settings

We can inject the value of the portal container settings into the portal container configuration files thanks to the variables which name start with "*portal.container.*", so to get the value of a setting called "*foo*", just use the following syntax *${portal.container.foo}*. You can also use internal variables, such as:

Table 53.4. Definition of the internal variables

portal.container.name	Gives the name of the current portal container.
portal.container.rest	Gives the context name of the rest web application of the current portal container.
portal.container.realm	

	Gives the realm name of the current portal container.

You can find below an example of how to use the variables:

```
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
  <component>
    <type>org.exoplatform.container.TestPortalContainer$MyComponent</type>
    <init-params>
      <!-- The name of the portal container -->
      <value-param>
        <name>portal</name>
        <value>${portal.container.name}</value>
      </value-param>
      <!-- The name of the rest ServletContext -->
      <value-param>
        <name>rest</name>
        <value>${portal.container.rest}</value>
      </value-param>
      <!-- The name of the realm -->
      <value-param>
        <name>realm</name>
        <value>${portal.container.realm}</value>
      </value-param>
      <value-param>
        <name>foo</name>
        <value>${portal.container.foo}</value>
      </value-param>
      <value-param>
        <name>before foo after</name>
        <value>before ${portal.container.foo} after</value>
      </value-param>
    </init-params>
  </component>
</configuration>
```

In the properties file corresponding to the external settings, you can reuse variables previously defined (in the external settings or in the internal settings) to create a new variable. In this case, the prefix "*portal.container.*" is not needed, see an example below:

```
my-var1=value 1
my-var2=value 2
complex-value=${my-var1}-${my-var2}
```

In the external and internal settings, you can also use create variables based on value of System paramaters. The System parameters can either be defined at launch time or thanks to the `PropertyConfigurator` (see next section for more details). See an example below:

```
temp-dir=${java.io.tmpdir}${file.separator}my-temp
```

However, for the internal settings, you can use System parameters only to define settings of type `java.lang.String`.

It cans be also very usefull to define a generic variable in the settings of the default portal container, the value of this variable will change according to the current portal container. See below an example:

```
my-generic-var=value of the portal container "${name}"
```

If this variable is defined at the default portal container level, the value of this variable for a portal container called *"foo"* will be *value of the portal container "foo"*.

Adding dynamically settings and/or dependencies to a `PortalContainer`

It is possible to use `component-plugin` elements in order to dynamically change a PortalContainerDefinition. In the example below, we add the dependency `foo` to the default portal container and to the portal containers called `foo1` and `foo2`:

```
<external-component-plugins>
  <!-- The full qualified name of the PortalContainerConfig -->
            <target-component>org.exoplatform.container.definition.PortalContainerConfig</target-
component>
  <component-plugin>
    <!-- The name of the plugin -->
    <name>Change PortalContainer Definitions</name>
   <!-- The name of the method to call on the PortalContainerConfig in order to register the changes
 on the PortalContainerDefinitions -->
    <set-method>registerChangePlugin</set-method>
    <!-- The full qualified name of the PortalContainerDefinitionChangePlugin -->
    <type>org.exoplatform.container.definition.PortalContainerDefinitionChangePlugin</type>
    <init-params>
```

```
    <value-param>
      <name>apply.default</name>
      <value>true</value>
    </value-param>
    <values-param>
      <name>apply.specific</name>
      <value>foo1</value>
      <value>foo2</value>
    </values-param>
    <object-param>
      <name>change</name>
                                                                    <object
type="org.exoplatform.container.definition.PortalContainerDefinitionChange$AddDependencies">
        <!-- The list of name of the dependencies to add -->
        <field name="dependencies">
          <collection type="java.util.ArrayList">
            <value>
              <string>foo</string>
            </value>
          </collection>
        </field>
      </object>
    </object-param>
  </init-params>
 </component-plugin>
</external-component-plugins>
```

Table 53.5. Descriptions of the fields of a `PortalContainerDefinitionChangePlugin`

apply.all (*)	Indicates whether the changes have to be applied to all the portal containers or not. The default value of this field is `false`. This field is a `ValueParam` and is not mandatory.
apply.default (*)	Indicates whether the changes have to be applied to the default portal container or not. The default value of this field is `false`. This field is a `ValueParam` and is not mandatory.
apply.specific (*)	A set of specific portal container names to which we want to apply the changes. This field is a `ValuesParam` and is not mandatory.
`Rest of the expected parameters`	The rest of the expected paramaters are `ObjectParam` of type

	`PortalContainerDefinitionChange`. Those parameters are in fact the list of changes that we want to apply to one or several portal containers. If the list of changes is empty, the component plugin will be ignored. The supported implementations of PortalContainerDefinitionChange are described later in this section.

Note

All the value of the parameters marked with a (*) can be defined thanks to System properties like any values in configuration files but also thanks to variables loaded by the *PropertyConfigurator*. For example in GateIn by default, it would be all the variables defined in the file *configuration.properties*.

To identify the portal containers to which the changes have to be applied, we use the follwing algorithm:

1. The parameter `apply.all` has been set to `true`. The corresponding changes will be applied to all the portal containers. The other parameters will be ignored.
2. The parameter `apply.default` has been set to `true` and the parameter `apply.specific` is `null`. The corresponding changes will be applied to the default portal container only.
3. The parameter `apply.default` has been set to `true` and the parameter `apply.specific` is not `null`. The corresponding changes will be applied to the default portal container and the given list of specific portal containers.
4. The parameter `apply.default` has been set to `false` or has not been set and the parameter `apply.specific` is `null`. The corresponding changes will be applied to the default portal container only.
5. The parameter `apply.default` has been set to `false` or has not been set and the parameter `apply.specific` is not `null`. The corresponding changes will be applied to the given list of specific portal containers.

The existing implementations of `PortalContainerDefinitionChange`

The modifications that can be applied to a `PortalContainerDefinition` must be a class of type `PortalContainerDefinitionChange`. The product proposes out of the box some implementations that we describe in the next sub sections.

`AddDependencies`

This modification adds a list of dependencies at the end of the list of dependencies defined into the `PortalContainerDefinition`. The full qualified name is *org.exoplatform.container.definition.PortalContainerDefinitionChange$AddDependencies*.

Table 53.6. Descriptions of the fields of an `AddDependencies`

dependencies	A list of *String* corresponding to the list of name of the dependencies to add. If the value of this field is empty, the change will be ignored.

See an example below, that will add `foo` at the end of the dependency list of the default portal container:

```
<external-component-plugins>
  <!-- The full qualified name of the PortalContainerConfig -->
            <target-component>org.exoplatform.container.definition.PortalContainerConfig</target-
component>
  <component-plugin>
    <!-- The name of the plugin -->
    <name>Change PortalContainer Definitions</name>
   <!-- The name of the method to call on the PortalContainerConfig in order to register the changes
 on the PortalContainerDefinitions -->
    <set-method>registerChangePlugin</set-method>
    <!-- The full qualified name of the PortalContainerDefinitionChangePlugin -->
    <type>org.exoplatform.container.definition.PortalContainerDefinitionChangePlugin</type>
    <init-params>
      <value-param>
        <name>apply.default</name>
        <value>true</value>
      </value-param>
      <object-param>
        <name>change</name>
                                                                                <object
 type="org.exoplatform.container.definition.PortalContainerDefinitionChange$AddDependencies">
          <!-- The list of name of the dependencies to add -->
          <field name="dependencies">
            <collection type="java.util.ArrayList">
              <value>
                <string>foo</string>
              </value>
            </collection>
          </field>
        </object>
      </object-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

AddDependenciesBefore

This modification adds a list of dependencies before a given target dependency defined into the list of dependencies of the `PortalContainerDefinition`. The full qualified name is *org.exoplatform.container.definition.PortalContainerDefinitionChange$AddDependenciesBefore*.

Table 53.7. Descriptions of the fields of an AddDependenciesBefore

dependencies	A list of *String* corresponding to the list of name of the dependencies to add. If the value of this field is empty, the change will be ignored.
target	The name of the dependency before which we would like to add the new dependencies. If this field is `null` or the target dependency cannot be found in the list of dependencies defined into the `PortalContainerDefinition`, the new dependencies will be added in first position to the list.

See an example below, that will add `foo` before `foo2` in the dependency list of the default portal container:

```
<external-component-plugins>
  <!-- The full qualified name of the PortalContainerConfig -->
            <target-component>org.exoplatform.container.definition.PortalContainerConfig</target-
component>
  <component-plugin>
    <!-- The name of the plugin -->
    <name>Change PortalContainer Definitions</name>
   <!-- The name of the method to call on the PortalContainerConfig in order to register the changes
 on the PortalContainerDefinitions -->
    <set-method>registerChangePlugin</set-method>
    <!-- The full qualified name of the PortalContainerDefinitionChangePlugin -->
    <type>org.exoplatform.container.definition.PortalContainerDefinitionChangePlugin</type>
    <init-params>
      <value-param>
        <name>apply.default</name>
        <value>true</value>
      </value-param>
      <object-param>
        <name>change</name>
                                                                                <object
 type="org.exoplatform.container.definition.PortalContainerDefinitionChange$AddDependenciesBefore">
          <!-- The list of name of the dependencies to add -->
          <field name="dependencies">
```

```
          <collection type="java.util.ArrayList">
            <value>
              <string>foo</string>
            </value>
          </collection>
        </field>
        <!-- The name of the target dependency -->
        <field name="target">
          <string>foo2</string>
        </field>
      </object>
    </object-param>
  </init-params>
 </component-plugin>
</external-component-plugins>
```

`AddDependenciesAfter`

This modification adds a list of dependencies before a given target dependency defined into the list of dependencies of the `PortalContainerDefinition`. The full qualified name is *org.exoplatform.container.definition.PortalContainerDefinitionChange$AddDependenciesAfter*.

Table 53.8. Descriptions of the fields of an `AddDependenciesAfter`

dependencies	A list of *String* corresponding to the list of name of the dependencies to add. If the value of this field is empty, the change will be ignored.
target	The name of the dependency after which we would like to add the new dependencies. If this field is `null` or the target dependency cannot be found in the list of dependencies defined into the `PortalContainerDefinition`, the new dependencies will be added in last position to the list.

See an example below, that will add `foo` after `foo2` in the dependency list of the default portal container:

```
<external-component-plugins>
 <!-- The full qualified name of the PortalContainerConfig -->
          <target-component>org.exoplatform.container.definition.PortalContainerConfig</target-
component>
 <component-plugin>
  <!-- The name of the plugin -->
```

```
    <name>Change PortalContainer Definitions</name>
   <!-- The name of the method to call on the PortalContainerConfig in order to register the changes
 on the PortalContainerDefinitions -->
    <set-method>registerChangePlugin</set-method>
    <!-- The full qualified name of the PortalContainerDefinitionChangePlugin -->
    <type>org.exoplatform.container.definition.PortalContainerDefinitionChangePlugin</type>
    <init-params>
     <value-param>
       <name>apply.default</name>
       <value>true</value>
     </value-param>
     <object-param>
       <name>change</name>
                                                                                <object
 type="org.exoplatform.container.definition.PortalContainerDefinitionChange$AddDependenciesAfter">
         <!-- The list of name of the dependencies to add -->
         <field name="dependencies">
           <collection type="java.util.ArrayList">
             <value>
               <string>foo</string>
             </value>
           </collection>
         </field>
         <!-- The name of the target dependency -->
         <field name="target">
           <string>foo2</string>
         </field>
       </object>
     </object-param>
   </init-params>
  </component-plugin>
</external-component-plugins>
```

`AddSettings`

This modification adds new settings to a `PortalContainerDefinition`. The full qualified name is *org.exoplatform.container.definition.PortalContainerDefinitionChange$AddSettings*.

Table 53.9. Descriptions of the fields of an `AddSettings`

settings	A map of <*String, Object*> corresponding to the settings to add. If the value of this field is empty, the change will be ignored.

See an example below, that will add the settings `string` and `stringX` to the settings of the default portal container:

```
<external-component-plugins>
  <!-- The full qualified name of the PortalContainerConfig -->
            <target-component>org.exoplatform.container.definition.PortalContainerConfig</target-
component>
  <component-plugin>
    <!-- The name of the plugin -->
    <name>Change PortalContainer Definitions</name>
   <!-- The name of the method to call on the PortalContainerConfig in order to register the changes
 on the PortalContainerDefinitions -->
    <set-method>registerChangePlugin</set-method>
    <!-- The full qualified name of the PortalContainerDefinitionChangePlugin -->
    <type>org.exoplatform.container.definition.PortalContainerDefinitionChangePlugin</type>
    <init-params>
      <value-param>
        <name>apply.default</name>
        <value>true</value>
      </value-param>
      <object-param>
        <name>change</name>
                                                                                <object
 type="org.exoplatform.container.definition.PortalContainerDefinitionChange$AddSettings">
          <!-- The settings to add to the to the portal containers -->
          <field name="settings">
            <map type="java.util.HashMap">
              <entry>
                <key>
                  <string>string</string>
                </key>
                <value>
                  <string>value1</string>
                </value>
              </entry>
              <entry>
                <key>
                  <string>stringX</string>
                </key>
                <value>
                  <string>value1</string>
                </value>
              </entry>
            </map>
```

```
        </field>
       </object>
     </object-param>
   </init-params>
  </component-plugin>
</external-component-plugins>
```

Disable dynamically a portal container

It is possible to use `component-plugin` elements in order to dynamically disable one or several portal containers. In the example below, we disable the portal container named `foo`:

```
<external-component-plugins>
  <!-- The full qualified name of the PortalContainerConfig -->
            <target-component>org.exoplatform.container.definition.PortalContainerConfig</target-component>
  <component-plugin>
    <!-- The name of the plugin -->
    <name>Disable a PortalContainer</name>
   <!-- The name of the method to call on the PortalContainerConfig in order to register the changes on the PortalContainerDefinitions -->
    <set-method>registerDisablePlugin</set-method>
    <!-- The full qualified name of the PortalContainerDefinitionDisablePlugin -->
    <type>org.exoplatform.container.definition.PortalContainerDefinitionDisablePlugin</type>
    <init-params>
      <!-- The list of the name of the portal containers to disable -->
      <values-param>
        <name>names</name>
        <value>foo</value>
      </values-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

Table 53.10. Descriptions of the fields of a `PortalContainerDefinitionDisablePlugin`

names (*)	The list of the name of the portal containers to disable.

Note

All the value of the parameters marked with a (*) can be defined thanks to System properties like any values in configuration files but also thanks to variables loaded by the *PropertyConfigurator*. For example in GateIn by default, it would be all the variables defined in the file *configuration.properties*.

To prevent any accesses to a web application corresponding to `PortalContainer` that has been disabled, you need to make sure that the following Http Filter (or a sub class of it) has been added to your web.xml in first position as below:

```
<filter>
  <filter-name>PortalContainerFilter</filter-name>
  <filter-class>org.exoplatform.container.web.PortalContainerFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>PortalContainerFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Note

It is only possible to disable a portal container when at least one PortalContainerDefinition has been registered.

System property configuration

A new property configurator service has been developed for taking care of configuring system properties from the inline kernel configuration or from specified property files.

The services is scoped at the root container level because it is used by all the services in the different portal containers in the application runtime.

Properties init param

The properties init param takes a property declared to configure various properties.

```
<component>
  <key>PropertyManagerConfigurator</key>
  <type>org.exoplatform.container.PropertyConfigurator</type>
  <init-params>
    <properties-param>
      <name>properties</name>
```

```
      <property name="foo" value="bar"/>
    </properties-param>
  </init-params>
</component>
```

Properties URL init param

The properties URL init param allow to load an external file by specifying its URL. Both property and XML format are supported, see the javadoc of the `java.util.Properties` class for more information. When a property file is loaded the various property declarations are loaded in the order in which the properties are declared sequentially in the file.

```
<component>
  <key>PropertyManagerConfigurator</key>
  <type>org.exoplatform.container.PropertyConfigurator</type>
  <init-params>
    <value-param>
      <name>properties.url</name>
      <value>classpath:configuration.properties</value>
    </value-param>
  </init-params>
</component>
```

In the properties file corresponding to the external properties, you can reuse variables before defining to create a new variable. In this case, the prefix "*portal.container.*" is not needed, see an example below:

```
my-var1=value 1
my-var2=value 2
complex-value=${my-var1}-${my-var2}
```

System Property configuration of the properties URL

It is possible to replace the properties URL init param by a system property that overwrites it. The name of that property is *exo.properties.url*.

Variable Syntaxes

All the variables that we described in the previous chapters can be defined thanks to 2 possible syntaxes which are *${variable-name}* or *${variable-name:default-value}*. The first syntax doesn't define any default value so if the variable has not be set the value will be *${variable-name}* to

indicate that it could not be resolved. The second syntax allows you to define the default value after the semi colon so if the variable has not be set the value will be the given default value.

Runtime configuration profiles

The kernel configuration is able to handle configuration profiles at runtime (as opposed to packaging time).

Profiles activation

An active profile list is obtained during the boot of the root container and is composed of the system property *exo.profiles* sliced according the "," delimiter and also a server specific profile value (tomcat for tomcat, jboss for jboss, etc...).

```
# runs GateIn on Tomcat with the profiles tomcat and foo
sh gatein.sh -Dexo.profiles=foo

# runs GateIn on JBoss with the profiles jboss, foo and bar
sh run.sh -Dexo.profiles=foo,bar
```

Profiles configuration

Profiles are configured in the configuration files of the eXo kernel.

Profiles definition

Profile activation occurs at XML to configuration object unmarshalling time. It is based on an "profile" attribute that is present on some of the XML element of the configuration files. To enable this, the kernel configuration schema has been upgraded to kernel_1_1.xsd. The configuration is based on the following rules:

1. Any kernel element with the no *profiles* attribute will create a configuration object
2. Any kernel element having a *profiles* attribute containing at least one of the active profiles will create a configuration object
3. Any kernel element having a *profiles* attribute matching none of the active profile will not create a configuration object
4. Resolution of duplicates (such as two components with same type) is left up to the kernel

Profiles capable configuration elements

A configuration element is *profiles* capable when it carries a profiles element.

Component element

The component element declares a component when activated. It will shadow any element with the same key declared before in the same configuration file:

```
<component>
  <key>Component</key>
  <type>Component</type>
</component>

<component profiles="foo">
  <key>Component</key>
  <type>FooComponent</type>
</component>
```

Component plugin element

The component-plugin element is used to dynamically extend the configuration of a given component. Thanks to the profiles the component-plugins could be enabled or disabled:

```
<external-component-plugins>
  <target-component>Component</target-component>
  <component-plugin profiles="foo">
    <name>foo</name>
    <set-method>addPlugin</set-method>
    <type>type</type>
    <init-params>
      <value-param>
        <name>param</name>
        <value>empty</value>
      </value-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

Import element

The import element imports a referenced configuration file when activated:

```
<import>empty</import>
<import profiles="foo">foo</import>
```

```
<import profiles="bar">bar</import>
```

Init param element

The init param element configures the parameter argument of the construction of a component service:

```
<component>
  <key>Component</key>
  <type>ComponentImpl</type>
  <init-params>
    <value-param>
      <name>param</name>
      <value>empty</value>
    </value-param>
    <value-param profiles="foo">
      <name>param</name>
      <value>foo</value>
    </value-param>
    <value-param profiles="bar">
      <name>param</name>
      <value>bar</value>
    </value-param>
  </init-params>
</component>
```

Value collection element

The value collection element configures one of the value of collection data:

```
<object type="org.exoplatform.container.configuration.ConfigParam">
  <field name="role">
    <collection type="java.util.ArrayList">
      <value><string>manager</string></value>
      <value profiles="foo"><string>foo_manager</string></value>
      <value profiles="foo,bar"><string>foo_bar_manager</string></value>
    </collection>
  </field>
</object>
```

Field configuration element

The field configuration element configures the field of an object:

```
<object-param>
  <name>test.configuration</name>
  <object type="org.exoplatform.container.configuration.ConfigParam">
    <field name="role">
      <collection type="java.util.ArrayList">
        <value><string>manager</string></value>
      </collection>
    </field>
    <field name="role" profiles="foo,bar">
      <collection type="java.util.ArrayList">
        <value><string>foo_bar_manager</string></value>
      </collection>
    </field>
    <field name="role" profiles="foo">
      <collection type="java.util.ArrayList">
        <value><string>foo_manager</string></value>
      </collection>
    </field>
  </object>
</object-param>
```

Component request life cycle

Component request life cycle contract

The component request life cycle is an interface that defines a contract for a component for being involved into a request:

```
public interface ComponentRequestLifecycle
{
   /**
    * Start a request.
    * @param container the related container
    */
   void startRequest(ExoContainer container);

   /**
    * Ends a request.
    * @param container the related container
```

```
   */
  void endRequest(ExoContainer container);
}
```

The container passed is the container to which the component is related. This contract is often used to setup a thread local based context that will be demarcated by a request.

For instance in the GateIn portal context, a component request life cycle is triggered for user requests. Another example is the initial data import in GateIn that demarcates using callbacks made to that interface.

Request life cycle

The `RequestLifeCycle` class has several statics methods that are used to schedule the component request life cycle of components. Its main responsability is to perform scheduling while respecting the constraint to execute the request life cycle of a component only once even if it can be scheduled several times.

Scheduling a component request life cycle

```
RequestLifeCycle.begin(component);
try
{
  // Do something
}
finally
{
  RequestLifeCycle.end();
}
```

Scheduling a container request life cycle

Scheduling a container triggers the component request life cyle of all the components that implement the interface `ComponentRequestLifeCycle`. If one of the component has already been scheduled before and then that component will not be scheduled again. When the local value is true, then the looked components will be those of the container, when it is false then the scheduler will also look at the components in the ancestor containers.

```
RequestLifeCycle.begin(container, local);
try
{
  // Do something
}
```

```
finally
{
  RequestLifeCycle.end();
}
```

When request life cycle is triggered

Portal request life cycle

Each portal request triggers the life cycle of the associated portal container.

JMX request Life Cycle

When a JMX bean is invoked, the request life cycle of the container to which it belongs it scheduled. Indeed JMX is an entry point of the system that may need component to have a request life cycle triggered.

Inversion Of Control

Overview

The services are not responsible for the instantiation of the components on which they depend.

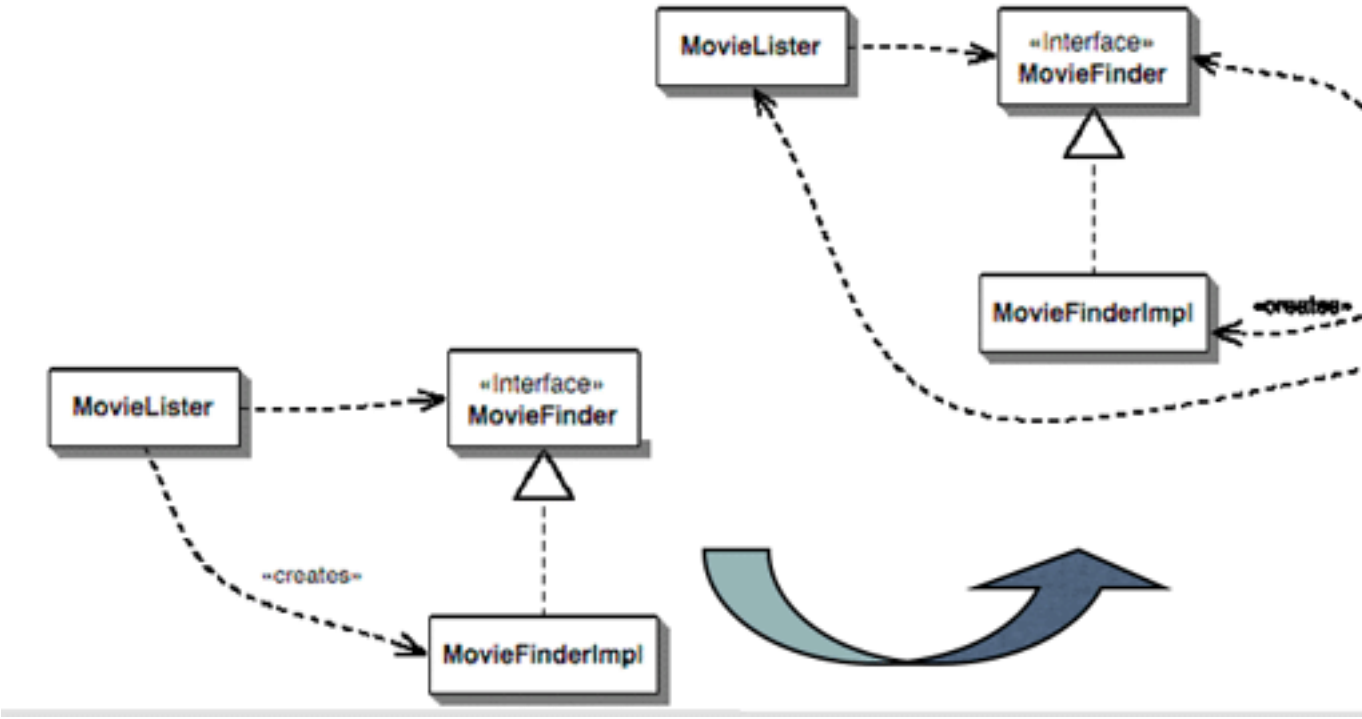
This architecture provides a loosely coupled design where the implementation of dependant services can be transparently exchanged.

This pattern has several names :

- Hollywood principle : "don't call me, I will call you"
- Inversion of Control
- Dependency injection

How

Don't let the object create itself the instances of the object that it references. This job is delegated to the container (assembler in the picture).



Injection

There are two ways to inject a dependency :

Using a constructor:

```
public ServiceA(ServiceB serviceB)
```

Using setter methods:

```
public void setServiceB(ServiceB serviceB)
```

When a client service can not be stored in the container then the service locator pattern is used:

```
public ServiceA(){
  this.serviceB =Container.getSInstance().getService(ServiceB.class);
}
```

Side effects

- Ease Unit test (use of Mock objects)
- Ease Maintainability
- Ease Refactoring
- Component reuse (POJOs != EJBs)

Services Wiring

Overview

The container package is responsible of building a hierarchy of containers. Each service will then be registered in one container or the other according to the XML configuration file it is defined in. It is important to understand that there can be several PortalContainer instances that all are children of the RootContainer.

The behavior of the hierarchy is similar to a class loader one, hence when you will lookup a service that depends on another one, the container will look for it in the current container and if it cannot be found, then it will look in the parent container. That way you can load all the reusable business logic components in the same container (here the RootContainer) and differentiate the service implementation from one portal instance to the other by just loading different service implementations in two sibling PortalContainers.

Therefore, if you look at the Portal Container as a service repository for all the business logic in a portal instance, then you understand why several PortalContainers allows you to manage several portals (each one deployed as a single war) in the same server by just changing XML configuration files.

The default configuration XML files are packaged in the service jar. There are three configuration.xml files, one for each container type. In that XML file, we define the list of services and their init parameters that will be loaded in the corresponding container.

Portal Instance

As there can be several portal container instances per JVM. it is important to be able to configure the loaded services per instance. Therefore all the default configuration files located in the service impl jar can be overridden from the portal war. For more information refer to *Service Configuration for Beginners*.

Introduction to the XML schema of the configuration.xml file

After deploying you find the configuration.xml file in webapps/portal/WEB-INF/conf Use component registration tags. Let's look at the key tag that defines the interface and the type tag that defines the implementation. Note that the key tag is not mandatory, but it improves performance.

```
<!-- Portlet container hooks -->
  <component>
   <key>org.exoplatform.services.portletcontainer.persistence.PortletPreferencesPersister</key>
    <type>org.exoplatform.services.portal.impl.PortletPreferencesPersisterImpl</type>
```

```
  </component>
```

Register plugins that can act as listeners or external plugin to bundle some plugin classes in other jar modules. The usual example is the hibernate service to which we can add hbm mapping files even if those are deployed in an other maven artifact.

```
<external-component-plugins>
  <target-component>org.exoplatform.services.database.HibernateService</target-component>
  <component-plugin>
    <name>add.hibernate.mapping</name>
    <set-method>addPlugin</set-method>
    <type>org.exoplatform.services.database.impl.AddHibernateMappingPlugin</type>
    <init-params>
      <values-param>
        <name>hibernate.mapping</name>
        <value>org/exoplatform/services/portal/impl/PortalConfigData.hbm.xml</value>
        <value>org/exoplatform/services/portal/impl/PageData.hbm.xml</value>
        <value>org/exoplatform/services/portal/impl/NodeNavigationData.hbm.xml</value>
      </values-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

In that sample we target the HibernateService and we will call its addPlugin() method with an argument of the type AddHibernateMappingPlugin. That object will first have been filled with the init parameters.

Therefore, it is possible to define services that will be able to receive plugins without implementing any framework interface.

Another example of use is the case of listeners as in the following code where a listener is added to the OrganisationService and will be called each time a new user is created:

```
<external-component-plugins>
          <target-component>org.exoplatform.services.organization.OrganizationService</target-
component>
  <component-plugin>
    <name>portal.new.user.event.listener</name>
    <set-method>addListenerPlugin</set-method>
    <type>org.exoplatform.services.portal.impl.PortalUserEventListenerImpl</type>
    <description>this listener create the portal configuration for the new user</description>
    <init-params>
      <object-param>
```

```
      <name>configuration</name>
      <description>description</description>
      <object type="org.exoplatform.services.portal.impl.NewPortalConfig">
        <field  name="predefinedUser">
          <collection type="java.util.HashSet">
            <value><string>admin</string></value>
            <value><string>exo</string></value>
            <value><string>company</string></value>
            <value><string>community</string></value>
            <value><string>portal</string></value>
            <value><string>exotest</string></value>
          </collection>
        </field>
        <field  name="templateUser"><string>template</string></field>
        <field  name="templateLocation"><string>war:/conf/users</string></field>
      </object>
    </object-param>
  </init-params>
 </component-plugin>
...
```

In the previous XML configuration, we refer the organization service and we will call its method addListenerPlugin with an object of type PortalUserEventListenerImpl. Each time a new user will be created (apart the predefined ones in the list above) methods of the PortalUserEventListenerImpl will be called by the service.

As you can see, there are several types of init parameters, from a simple value param which binds a key with a value to a more complex object mapping that fills a JavaBean with the info defined in the XML.

Many other examples exist such as for the Scheduler Service where you can add a job with a simple XML configuration or the JCR Service where you can add a NodeType from your own configuration.xml file.

Configuration retrieval and log of this retrieval

When the RootContainer is starting the configuration retrieval looks for configuration files in each jar available from the classpath at jar path /conf/portal/configuration.xml and from each war at path /WEB-INF/conf/configuration.xml. These configurations are added to a set. If a component was configured in a previous jar and the current jar contains a new configuration of that component the latest (from the current jar) will replace the previous configuration.

After the processing of all configurations available on the system the container will initialize it and start each component in order of the dependency injection (DI).

So, in general the user/developer should be careful when configuring the same components in different configuration files. It's recommended to configure service in its own jar only. Or, in case of a portal configuration, strictly reconfigure the component in portal files.

But, there are components that can be (or should be) configured more than one time. This depends on the business logic of the component. A component may initialize the same resource (shared with other players) or may add a particular object to a set of objects (shared with other players too). In the first case it's critical who will be the last, i.e. whose configuration will be used. In second case it doesn't matter who is the first and who is the last (if the parameter objects are independent).

In case of problems with configuration of component it's important to know from which jar/war it comes. For that purpose user/developer can set JVM system property **org.exoplatform.container.configuration.debug**, in command line:

```
java -Dorg.exoplatform.container.configuration.debug ...
```

With that property container configuration manager will report configuration adding process to the standard output (System.out).

```
  ......
  Add configuration jar:file:/D:/Projects/eXo/dev/exo-working/exo-tomcat/lib/exo.kernel.container-trunk.jar!/conf/portal/configuration.xml
                Add configuration jar:file:/D:/Projects/eXo/dev/exo-working/exo-tomcat/lib/exo.kernel.component.cache-trunk.jar!/conf/portal/configuration.xml
  Add configuration jndi:/localhost/portal/WEB-INF/conf/configuration.xml
      import jndi:/localhost/portal/WEB-INF/conf/common/common-configuration.xml
      import jndi:/localhost/portal/WEB-INF/conf/database/database-configuration.xml
      import jndi:/localhost/portal/WEB-INF/conf/ecm/jcr-component-plugins-configuration.xml
      import jndi:/localhost/portal/WEB-INF/conf/jcr/jcr-configuration.xml
  ......
```

Component Plugin Priority

Since kernel version 2.0.6 it is possible to setup order of loading for ComponentPlugin. Use the ' **priority**' tag to define plugin's load priority. By **default** all plugins get **priority '0'**; they will be loaded in the container's natural way. If you want one plugin to be loaded later than the others then just set priority for it **higher than zero**.

Simple example of fragment of a **configuration.xml**.

```
...
<component>
  <type>org.exoplatform.services.Component1</type>
</component>

<external-component-plugins>
  <target-component>org.exoplatform.services.Component1</target-component>

  <component-plugin>
    <name>Plugin1</name>
    <set-method>addPlugin</set-method>
    <type>org.exoplatform.services.plugins.Plugin1</type>
    <description>description</description>
    <priority>1</priority>
  </component-plugin>

  <component-plugin>
    <name>Plugin2</name>
    <set-method>addPlugin</set-method>
    <type>org.exoplatform.services.plugins.Plugin2</type>
    <description>description</description>
    <priority>2</priority>
  </component-plugin>

</external-component-plugins>

<external-component-plugins>
  <target-component>org.exoplatform.services.Component1</target-component>
  <component-plugin>
    <name>Plugin3</name>
    <set-method>addPlugin</set-method>
    <type>org.exoplatform.services.plugins.Plugin3</type>
    <description>description</description>
  </component-plugin>
```

```
</external-component-plugins>
...
```

In the above example plugin 'Plugin3' will be loaded first because it has the default priority '0'. Then, plugin 'Plugin1' will be loaded and last one is plugin 'Plugin2'.

Understanding the ListenerService

Related documents

- *Service Configuration for Beginners*
- *Service Configuration in Detail*
- *Container Configuration*

Objectives

This article will first describe how the ListenerService works and then it will show you how to configure the ListenerService.

What is the ListenerService ?

Inside eXo, an event mechanism allows to trigger and listen to events under specific conditions. This mechanism is used in several places in eXo such as login/logout time.

How does it work?

Listeners must be subclasses of org.exoplatform.services.listener.Listener registered by the ListenerService.

Registering a listener

To register a listener, you need to call the addListener() method.

```
/**
 * This method is used to register a listener with the service. The method
 * should: 1. Check to see if there is a list of listener with the listener
 * name, create one if the listener list doesn't exit 2. Add the new listener
 * to the listener list
 *
 * @param listener
*/
public void addListener(Listener listener) {
   ...
}
```

By convention, we use the listener name as the name of the event to listen to.

Triggering an event

To trigger an event, an application can call one of the broadcast() methods of ListenerService.

```
/**
 * This method is used to broadcast an event. This method should: 1. Check if
 * there is a list of listener that listen to the event name. 2. If there is a
 * list of listener, create the event object with the given name , source and
 * data 3. For each listener in the listener list, invoke the method
 * onEvent(Event)
 *
 * @param <S> The type of the source that broadcast the event
 * @param <D> The type of the data that the source object is working on
 * @param name The name of the event
 * @param source The source object instance
 * @param data The data object instance
 * @throws Exception
 */
public <S, D> void broadcast(String name, S source, D data) throws Exception {
   ...
}

/**
 * This method is used when a developer want to implement his own event object
 * and broadcast the event. The method should: 1. Check if there is a list of
 * listener that listen to the event name. 2. If there is a list of the
 * listener, For each listener in the listener list, invoke the method
 * onEvent(Event)
 *
 * @param <T> The type of the event object, the type of the event object has
 *          to be extended from the Event type
 * @param event The event instance
 * @throws Exception
 */
public <T extends Event> void broadcast(T event) throws Exception {
   ...
}
```

The boadcast() methods retrieve the name of the event and find the registered listeners with the same name and call the method onEvent() on each listener found.

Each listener is a class that extends org.exoplatform.services.listener.Listener, as you can see below:

```
public abstract class Listener<S, D> extends BaseComponentPlugin {

  /**
   * This method should be invoked when an event with the same name is
   * broadcasted
   */
  public abstract void onEvent(Event<S, D> event) throws Exception;
}
```

> As you can see we use generics to limit the source of the event to the type 'S' and the data of the event to the type 'D', so we expect that listeners implement the method onEvent() with the corresponding types

Each listener is also a ComponentPlugin with a name and a description, in other words, the name of the listener will be the name given in the configuration file, for more details see the next section.

```
public interface ComponentPlugin {
  public String getName();

  public void setName(String name);

  public String getDescription();

  public void setDescription(String description);
}
```

How to configure a listener?

All listeners are in fact a ComponentPlugin so it must be configured as below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>
...
  <external-component-plugins>
    <!-- The full qualified name of the ListenerService -->
    <target-component>org.exoplatform.services.listener.ListenerService</target-component>

    <component-plugin>
    <!-- The name of the listener that is also the name of the target event -->
```

```
    <name>${name-of-the-target-event}</name>
    <!-- The name of the method to call on the ListenerService in order to register the Listener -->
    <set-method>addListener</set-method>
    <!-- The full qualified name of the Listener -->
    <type>${the-FQN-of-the-listener}</type>
  </component-plugin>

 </external-component-plugins>
</configuration>
```

Concrete Example

The org.exoplatform.services.security.ConversationRegistry uses the ListenerService to notify that a user has just signed in or just left the application. For example, when a new user signs in, the following code is called:

```
listenerService.broadcast("exo.core.security.ConversationRegistry.register", this, state);
```

This code will in fact create a new Event which name is "exo.core.security.ConversationRegistry.register", which source is the current instance of ConversationRegistry and which data is the given state. The ListenerService will call the method onEvent(Event<ConversationRegistry, ConversationState> event) on all the listeners which name is "exo.core.security.ConversationRegistry.register".

In the example below, we define a Listener that will listen the event "exo.core.security.ConversationRegistry.register".

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>
...
 <external-component-plugins>
  <!-- The full qualified name of the ListenerService -->
  <target-component>org.exoplatform.services.listener.ListenerService</target-component>

  <component-plugin>
   <!-- The name of the listener that is also the name of the target event -->
   <name>exo.core.security.ConversationRegistry.register</name>
   <!-- The name of the method to call on the ListenerService in order to register the Listener -->
   <set-method>addListener</set-method>
   <!-- The full qualified name of the Listener -->
   <type>org.exoplatform.forum.service.AuthenticationLoginListener</type>
  </component-plugin>
```

```
  </external-component-plugins>
</configuration>
...
```

Initial Context Binder

Initial Context Binder is responsible for binding references at runtime, persisting in file and automatically rebinding. Java temp directory is used to persist references in bind-references.xml file by default. In case when need to definde special file it can be done by add parameter to *InitialContextInitializer* configuration.

API

Service provide methods for binding reference:

```
public void bind(String bindName, String className, String factory, String factoryLocation, Map<String, String> refAddr) throws NamingException, FileNotFoundException, XMLStreamExcept
```

- bindName - name of binding
- className - the fully-qualified name of the class of the object to which this Reference refers
- factory - the name of the factory class for creating an instance of the object to which this Reference refers
- factoryLocation - the location of the factory class
- refAddr - object's properties map

```
public void bind(String bindName, Reference ref) throws NamingException, FileNotFoundException, XMLStreamExcept
```

Returns reference associated with defined name:

```
public Reference getReference(String bindName)
```

Unbind the Reference with defined name:

```
public void unbind(String bindName) throws NamingException, FileNotFoundException, XMLStreamException
```

Job Scheduler Service

What is Job Scheduler?

Job scheduler defines a job to execute a given number of times during a given period. It is a service that is in charge of unattended background executions, commonly known for historical reasons as batch processing. It is used to create and run jobs automatically and continuously, to schedule event-driven jobs and reports.

Where is Job Scheduler Service used in eXo Products?

Job Scheduler Service is widely used in many eXo products such as Social, DMS, WCM, eXo Knowledge and eXo Collaboration.

In eXo products, Job Schedulers are used to do some tasks as below:

- Automatically send notification, such as task/event reminder in the Calendar application of eXo Collaboration.
- Automatically save chat messages from Openfire Server to History in the Chat application of eXo Collaboration.
- Inactivate topics in the Forum application of eXo Knowledge.
- Calculate the number of active and online users in the Forum application of eXo Knowledge.
- Automatically collect RSS items from various RSS resources to post to the activity stream of users and spaces in eXo Social.
- Automatically send Newsletters to users in WCM.

Also, it is used in Schedule lifecycle in DMS.

By using Job Scheduler Service in eXo kernel, many kinds of job can be configured to run, such as, addPeriodJob, addCronJob, addGlobalJobListener, addJobListener and many more. Just write a job (a class implements Job interface of quartz library and configures plug-in for JobSchedulerService and you're done.

How does Job Scheduler work?

Jobs are scheduled to run when a given Trigger occurs. Triggers can be created with nearly any combination of the following directives:

- at a certain time of day (to the millisecond)
- on certain days of the week

- on certain days of the month
- on certain days of the year
- not on certain days listed within a registered Calendar (such as business holidays)
- repeated a specific number of times
- repeated until a specific time/date
- repeated indefinitely
- repeated with a delay interval

Jobs are given names by their creator and can also be organized into named groups. Triggers may also be given names and placed into groups, in order to easily organize them within the scheduler. Jobs can be added to the scheduler once, but registered with multiple Triggers. Within a J2EE environment, Jobs can perform their work as part of a distributed (XA) transaction.

(Source: quartz-scheduler.org)

How can Job Scheduler Service be used in Kernel?

Kernel leverages *Quartz* [http://www.quartz-scheduler.org] for its scheduler service and wraps `org.quartz.Scheduler` in `org.exoplatform.services.scheduler.impl.QuartzSheduler` for easier service wiring and configuration like any other services. To work with Quartz in Kernel, you will mostly work with `org.exoplatform.services.scheduler.JobSchedulerService` (implemented by `org.exoplatform.services.scheduler.impl.JobSchedulerServiceImpl`.

To use `JobSchedulerService`, you can configure it as a component in the configuration.xml. Because `JobSchedulerService` requires `QuartzSheduler` and `QueueTasks`, you also have to configure these two components.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd        http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">

  <component>
    <type>org.exoplatform.services.scheduler.impl.QuartzSheduler</type>
  </component>

  <component>
```

```
    <type>org.exoplatform.services.scheduler.QueueTasks</type>
  </component>

  <component>
    <key>org.exoplatform.services.scheduler.JobSchedulerService</key>
    <type>org.exoplatform.services.scheduler.impl.JobSchedulerServiceImpl</type>
  </component>

</configuration>
```

Samples

Note

You can download the project code from *here* [https://github.com/hoatle/job-scheduler-service-tutorial]

Work with `JobSchedulerService` by creating a sample project and use GateIn-3.1.0-GA for testing.

Firstly, create a project by using maven archetype plugin:

```
mvn archetype:generate
```

- For project type: select **maven-archetype-quickstart**
- For groupId: select **org.exoplatform.samples**
- For artifactId: select **exo.samples.scheduler**
- For version: select **1.0.0-SNAPSHOT**
- For package: select **org.exoplatform.samples.scheduler**

Edit the pom.xml as follows:

```
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0   http://maven.apache.org/maven-v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>
```

```
  <parent>
    <artifactId>exo.portal.parent</artifactId>
    <groupId>org.exoplatform.portal</groupId>
    <version>3.1.0-GA</version>
  </parent>

  <groupId>org.exoplatform.samples</groupId>
  <artifactId>exo.samples.scheduler</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <name>eXo Samples For Scheduler</name>
  <description>eXo Samples Code For Scheduler</description>
</project>
```

Generate an eclipse project by using maven eclipse plugin and then import into eclipse:

```
mvn eclipse:eclipse
```

eXo Kernel makes it easier to work with job scheduler service. All you need is just to define your "job" class to be performed by implementing *org.quartz.Job* interface and add configuration for it.

Define a job

To define a job, do as follows:

Define your job to be performed. For example, the job *DumbJob* is defined as follows:

```
package org.exoplatform.samples.scheduler.jobs;

import org.exoplatform.services.log.ExoLogger;
import org.exoplatform.services.log.Log;
import org.quartz.Job;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;

/**
 * DumbJob for executing a defined dumb job.
 */
public class DumbJob implements Job {

  /**
   * The logger
   */
  private static final Log LOG = ExoLogger.getLogger(DumbJob.class);
```

```
  /**
   * The job of the DumbJob will be done by executing this method.
   *
   * @param context
   * @throws JobExecutionException
   */
  public void execute(JobExecutionContext context) throws JobExecutionException {
    LOG.info("DumbJob is executing...");
  }
}
```

All jobs are required to implement the method *execute* from *org.quartz.Job* interface. This method will be called whenever a job is performed. With *DumbJob*, you just use logging to see that it will work. By looking at the terminal, you will see the the log message: "DumbJob is executing..."

Job configuration

After defining the "job", the only next step is to configure it by using *external-component-plugin* configuration for *org.exoplatform.services.scheduler.JobSchedulerService*. You can use these methods below for setting component plugin:

```
public void addPeriodJob(ComponentPlugin plugin) throws Exception;
```

The component plugin for this method must be the type of *org.exoplatform.services.scheduler.PeriodJob*. This type of job is used to perform actions that are executed in a period of time. You have to define when this job is performed, when it ends, when it performs the first action, how many times it is executed and the period of time to perform the action. See the configuration sample below to understand more clearly:

```
<external-component-plugins>
            <target-component>org.exoplatform.services.scheduler.JobSchedulerService</target-component>
   <component-plugin>
     <name>PeriodJob Plugin</name>
     <set-method>addPeriodJob</set-method>
     <type>org.exoplatform.services.scheduler.PeriodJob</type>
     <description>period job configuration</description>
     <init-params>
       <properties-param>
         <name>job.info</name>
         <description>dumb job executed  periodically</description>
```

```
        <property name="jobName" value="DumbJob"/>
        <property name="groupName" value="DumbJobGroup"/>
        <property name="job" value="org.exoplatform.samples.scheduler.jobs.DumbJob"/>
        <property name="repeatCount" value="0"/>
        <property name="period" value="60000"/>
        <property name="startTime" value="+45"/>
        <property name="endTime" value=""/>
      </properties-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

```
public void addCronJob(ComponentPlugin plugin) throws Exception;
```

The component plugin for this method must be the type of *org.exoplatform.services.scheduler.CronJob*. This type of job is used to perform actions at specified time with Unix 'cron-like' definitions. The plugin uses "expression" field for specifying the 'cron-like' definitions to execute the job. This is considered as the most powerful and flexible job to define when it will execute. For example, at 12pm every day => "0 0 12 * * ?"; or at 10:15am every Monday, Tuesday, Wednesday, Thursday and Friday => "0 15 10 ? * MON-FRI". To see more about Cron expression, please refer to this article:

CRON expression [http://en.wikipedia.org/wiki/CRON_expression].
See the configuration sample below to understand more clearly:

```
<external-component-plugins>
            <target-component>org.exoplatform.services.scheduler.JobSchedulerService</target-component>
  <component-plugin>
    <name>CronJob Plugin</name>
    <set-method>addCronJob</set-method>
    <type>org.exoplatform.services.scheduler.CronJob</type>
    <description>cron job configuration</description>
    <init-params>
      <properties-param>
        <name>job.info</name>
        <description>dumb job executed by cron expression</description>
        <property name="jobName" value="DumbJob"/>
        <property name="groupName" value="DumbJobGroup"/>
        <property name="job" value="org.exoplatform.samples.scheduler.jobs.DumbJob"/>
        <!-- The job will be performed at 10:15am every day -->
        <property name="expression" value="0 15 10 * * ?"/>
```

```
      </properties-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

```
public void addGlobalJobListener(ComponentPlugin plugin) throws Exception;
```

```
public void addJobListener(ComponentPlugin plugin) throws Exception;
```

The component plugin for two methods above must be the type of *org.quartz.JobListener.* This job listener is used so that it will be informed when a *org.quartz.JobDetail* executes.

```
public void addGlobalTriggerListener(ComponentPlugin plugin) throws Exception;
```

```
public void addTriggerListener(ComponentPlugin plugin) throws Exception;
```

The component plugin for two methods above must be the type of *org.quartz.TriggerListener*. This trigger listener is used so that it will be informed when a *org.quartz.Trigger* fires.

Run the project

Create *conf.portal* package in your sample project. Add the configuration.xml file with the content as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd          http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">

  <component>
    <type>org.exoplatform.services.scheduler.impl.QuartzSheduler</type>
  </component>
  <component>
    <type>org.exoplatform.services.scheduler.QueueTasks</type>
  </component>
  <component>
```

```
    <key>org.exoplatform.services.scheduler.JobSchedulerService</key>
    <type>org.exoplatform.services.scheduler.impl.JobSchedulerServiceImpl</type>
  </component>

  <external-component-plugins>
            <target-component>org.exoplatform.services.scheduler.JobSchedulerService</target-
component>
    <component-plugin>
      <name>PeriodJob Plugin</name>
      <set-method>addPeriodJob</set-method>
      <type>org.exoplatform.services.scheduler.PeriodJob</type>
      <description>period job configuration</description>
      <init-params>
        <properties-param>
          <name>job.info</name>
          <description>dumb job executed periodically</description>
          <property name="jobName" value="DumbJob"/>
          <property name="groupName" value="DumbJobGroup"/>
          <property name="job" value="org.exoplatform.samples.scheduler.jobs.DumbJob"/>
          <property name="repeatCount" value="0"/>
          <property name="period" value="60000"/>
          <property name="startTime" value="+45"/>
          <property name="endTime" value=""/>
        </properties-param>
      </init-params>
    </component-plugin>
  </external-component-plugins>
</configuration>
```

mvn clean install the project. Copy .jar file to *lib* in tomcat bundled with GateIn-3.1.0-GA. Run *bin/gatein.sh* to see the *DumbJob* to be executed on the terminal when portal containers are initialized. Please look at the terminal to see the log message of *DumbJob*.

From now on, you can easily create any job to be executed in GateIn's portal by defining your job and configuring it.

Reference

To further understand about Job Scheduler, you can refer the following links:

- *http://www.quartz-scheduler.org/*
- *http://en.wikipedia.org/wiki/Job_scheduler*
- *http://www.theserverside.com/news/1364726/Job-Scheduling-in-J2EE-Applications*

- *http://technet.microsoft.com/en-us/library/cc720070%28WS.10%29.aspx*

eXo Cache

Basic concepts

All applications on the top of eXo JCR that need a cache, can rely on an `org.exoplatform.services.cache.ExoCache` instance that is managed by the `org.exoplatform.services.cache.CacheService`. The main implementation of this service is `org.exoplatform.services.cache.impl.CacheServiceImpl` which depends on the `org.exoplatform.services.cache.ExoCacheConfig` in order to create new `ExoCache` instances. See the below example of `org.exoplatform.services.cache.CacheService` definition:

```
<component>
  <key>org.exoplatform.services.cache.CacheService</key>
  <jmx-name>cache:type=CacheService</jmx-name>
  <type>org.exoplatform.services.cache.impl.CacheServiceImpl</type>
  <init-params>
    <object-param>
      <name>cache.config.default</name>
      <description>The default cache configuration</description>
      <object type="org.exoplatform.services.cache.ExoCacheConfig">
        <field name="name"><string>default</string></field>
        <field name="maxSize"><int>300</int></field>
        <field name="liveTime"><long>600</long></field>
        <field name="distributed"><boolean>false</boolean></field>
                                                                <field

string></field>
      </object>
    </object-param>
  </init-params>
</component>
```

Note

The `ExoCacheConfig` which name is `default`, will be the default configuration of all the `ExoCache` instances that don't have dedicated configuration.

See the below example about how to define a new `ExoCacheConfig` thanks to a *external-component-plugin*:

```
<external-component-plugins>
```

```
  <target-component>org.exoplatform.services.cache.CacheService</target-component>
  <component-plugin>
    <name>addExoCacheConfig</name>
    <set-method>addExoCacheConfig</set-method>
    <type>org.exoplatform.services.cache.ExoCacheConfigPlugin</type>
    <description>Configures the cache for query service</description>
    <init-params>
      <object-param>
        <name>cache.config.wcm.composer</name>
        <description>The default cache configuration</description>
        <object type="org.exoplatform.services.cache.ExoCacheConfig">
          <field name="name"><string>wcm.composer</string></field>
          <field name="maxSize"><int>300</int></field>
          <field name="liveTime"><long>600</long></field>
          <field name="distributed"><boolean>false</boolean></field>
                                                                    <field

string></field>
        </object>
      </object-param>
    </init-params>
  </component-plugin>
 </external-component-plugins>
```

Table 60.1. Descriptions of the fields of `ExoCacheConfig`

name	The name of the cache. This field is mandatory since it will be used to retrieve the `ExoCacheConfig` corresponding to a given cache name.
label	The label of the cache. This field is optional. It is mainly used to indicate the purpose of the cache.
maxSize	The maximum numbers of elements in cache. This field is mandatory.
liveTime	The amount of time (in seconds) that an element is not written or read before it is evicted. This field is mandatory.
implementation	The full qualified name of the cache implementation to use. This field is optional. This field is only used for simple cache implementation. The default and

	main implementation is `org.exoplatform.services.cache.concurrent.ConcurrentFI` this implementation only works with local caches with FIFO as eviction policy. For more complex implementation see the next sections.
distributed	Indicates if the cache is distributed. This field is optional. This field is deprecated.
replicated	Indicates if the cache is replicated. This field is optional.
logEnabled	Indicates if the log is enabled. This field is optional. This field is used for backward compatibility.
avoidValueReplication	Indicates whether the values of the cache should be replicated or not in case of a replicated cache. This field is optional. By default it is disabled. Find more details about this field in the next section.

Advanced concepts

Invalidation

In case, you have big values or non serializable values and you need a replicated cache to at list invalidate the data when it is needed, you can use the invalidation mode that will work on top of any replicated cache implementations. This is possible thanks to the class *InvalidationExoCache* which is actually a decorator whose idea is to replicate the the hash code of the value in order to know if it is needed or not to invalidate the local data, if the new hash code of the value is the same as the old value, we assume that it is the same value so we don't invalidate the old value. This is required to avoid the following infinite loop that we will face with invalidation mode proposed out of the box by JBoss Cache for example:

1. Cluster node #1 puts (key1, value1) into the cache
2. On cluster node #2 key1 is invalidated by put call in node #1
3. Node #2 re-loads key1 and puts (key1, value1) into the cache
4. On cluster node #1 key1 is invalidated, so we get back to step #1

In the use case above, thanks to the *InvalidationExoCache* since the value loaded at step #3 has the same hash code as the value loaded as step #1, the step #4 won't invalidate the data on the cluster node #1.

It exists 2 ways to use the invalidation mode which are the following:

1. By configuration: For this you simply need to set the parameter *avoidValueReplication* to *true* in your eXo cache configuration, this will indicate the CacheService to wrap your eXo cache instance into an *InvalidationExoCache* in case the cache is defined as replicated or distributed.

2. Programmatically; You can wrap your eXo cache instance into an *org.exoplatform.services.cache.invalidation.InvalidationExoCache* yourself using the public constructors that are available. Please note that if you use *CacheListeners* add them to the InvalidationExoCache instance instead of the nested eXo Cache because the nested eXo Cache will contain only hash codes so the related listeners will get hash codes instead of the real values.

Note

The invalidation will be efficient if and only if the hash code method is properly implemented, in other words 2 value objects representing the same data need to return the same hash code otherwise the infinite loop described above will still be effective.

FutureExoCache

If the data that you want to store into your eXo Cache instance takes a lot of time to load and/or you would like to prevent multiple concurrent loading of the same data at the same time, you can use *org.exoplatform.services.cache.future.FutureExoCache* on top of your eXo Cache instance in order to delegate the loading of your data to a loader that will be called only once whatever the total amount of concurrent threads looking for it. See below an example of how the FutureExoCache can be used:

```
import org.exoplatform.services.cache.future.Loader;
import org.exoplatform.services.cache.future.FutureExoCache;
...
  // Define first your loader and choose properly your context object in order
  // to be able to reuse the same loader for different FutureExoCache instances
  Loader<String, String, String> loader = new Loader<String, String, String>()
  {
    public String retrieve(String context, String key) throws Exception
    {
      return "Value loaded thanks to the key = '" + key + "' and the context = '" + context + "'";
    }
  };
  // Create your FutureExoCache from your eXo cache instance and your loader
   FutureExoCache<String, String, String> myFutureExoCache = new FutureExoCache<String,
String, String>(loader, myExoCache);
  // Get your data from your future cache instance
  System.out.println(myFutureExoCache.get("my context", "foo"));
```

eXo Cache extension

In the previous versions of eXo kernel, it was quite complex to implement your own ExoCache because it was not open enough. Since kernel 2.0.8, it is possible to easily integrate your favorite cache provider in eXo Products.

You just need to implement your own `ExoCacheFactory` and register it in an eXo container, as described below:

```
package org.exoplatform.services.cache;
...
public interface ExoCacheFactory {

  /**
   * Creates a new instance of {@link org.exoplatform.services.cache.ExoCache}
   * @param config the cache to create
   * @return the new instance of {@link org.exoplatform.services.cache.ExoCache}
   * @exception ExoCacheInitException if an exception happens while initializing the cache
   */
  public ExoCache createCache(ExoCacheConfig config) throws ExoCacheInitException;
}
```

As you can see, there is only one method to implement which can be seen as a converter of an `ExoCacheConfig` to get an instance of `ExoCache`. Once, you created your own implementation, you can simply register your factory by adding a file *conf/portal/configuration.xml* with a content of the following type:

```
<configuration>
  <component>
    <key>org.exoplatform.services.cache.ExoCacheFactory</key>
    <type>org.exoplatform.tutorial.MyExoCacheFactoryImpl</type>
    ...
  </component>
</configuration>
```

Note

Since kernel 2.3.0-CR1, if the configuration is not a sub class of `ExoCacheConfig` and the implementation given in the configuration is the full qualified name of an existing implementation of eXo Cache, we will assume that the user expects to have an instance of this eXo Cache type so we won't use the configured cache factory.

eXo Cache based on JBoss Cache

Configuring the ExoCacheFactory

When you add, the eXo library in your classpath, the eXo service container will use the default configuration provided in the library itself but of course you can still redefined the configuration if you wish as you can do with any components.

The default configuration of the factory is:

```
<configuration>
  <component>
    <key>org.exoplatform.services.cache.ExoCacheFactory</key>
    <type>org.exoplatform.services.cache.impl.jboss.ExoCacheFactoryImpl</type>
    <init-params>
      <value-param>
        <name>cache.config.template</name>
        <value>jar:/conf/portal/cache-configuration-template.xml</value>
      </value-param>
      <value-param>
        <name>allow.shareable.cache</name>
        <value>true</value>
      </value-param>
    </init-params>
  </component>
</configuration>
```

Table 60.2. Fields description

cache.config.template	This parameter allows you to define the location of the default configuration template of JBoss Cache. In the default configuration, we ask the eXo kernel to get the file shipped into the jar at */conf/portal/cache-configuration-template.xml*. The default configuration template aims to be the skeleton from which we will create any type of jboss cache instance, thus it must be very generic. **Note** The default configuration template provided with the jar aims to work with any application servers, but if you

	intend to use JBoss AS, you should redefine it in your custom configuration to fit better with your AS.
allow.shareable.cache	This parameter allows you to Indicate whether the JBoss Cache instances used can by default be shared between several eXo caches instances. indeed to consume less resources, you can allow to use the same instance of JBoss Cache for several eXo Cache instances, each eXo Cache Instances will have his own cache region with its own eviction configuration. The default value of this parameter is *false*. **Note** This value is only the default value that cans be redefined at `ExoCacheConfig` level thanks to the field *allowShareableCache* for more details see the next sections.

Adding specific configuration for a cache

If for a given reason, you need to use a specific configuration for a cache, you can register one thanks to an "*external plugin*", see an example below:

```
<configuration>
  ...
  <external-component-plugins>
    <target-component>org.exoplatform.services.cache.ExoCacheFactory</target-component>
    <component-plugin>
      <name>addConfig</name>
      <set-method>addConfig</set-method>
      <type>org.exoplatform.services.cache.impl.jboss.ExoCacheFactoryConfigPlugin</type>
      <description>add Custom Configurations</description>
      <init-params>
        <value-param>
          <name>myCustomCache</name>
          <value>jar:/conf/portal/custom-cache-configuration.xml</value>
        </value-param>
      </init-params>
    </component-plugin>
  </external-component-plugins>
```

```
  ...
</configuration>
```

In the example above, I call the method *addConfig(ExoCacheFactoryConfigPlugin plugin)* on the current implementation of `ExoCacheFactory` which is actually the jboss cache implementation.

In the *init-params* block, you can define a set of *value-param* blocks and for each *value-param*, we expect the name of cache that needs a specific configuration as name and the location of your custom configuration as *value*.

In this example, we indicates to the factory that we would like that the cache *myCustomCache* use the configuration available at *jar:/conf/portal/custom-cache-configuration.xml*.

Adding a cache creator

Understanding a cache creator

The factory for jboss cache, delegates the cache creation to `ExoCacheCreator` that is defined as below:

```
package org.exoplatform.services.cache.impl.jboss;
...
public interface ExoCacheCreator {

  /**
         * Creates an eXo cache according to the given configuration {@link
 org.exoplatform.services.cache.ExoCacheConfig}
   * @param config the configuration of the cache to apply
   * @param cache the cache to initialize
   * @exception ExoCacheInitException if an exception happens while initializing the cache
   */
  public ExoCache create(ExoCacheConfig config, Cache<Serializable, Object> cache) throws
 ExoCacheInitException;

  /**
   * Returns the type of {@link org.exoplatform.services.cache.ExoCacheConfig} expected by the
 creator
   * @return the expected type
   */
  public Class<? extends ExoCacheConfig> getExpectedConfigType();

  /**
    * Returns the name of the implementation expected by the creator. This is mainly used to be
 backward compatible
```

```
   * @return the expected by the creator
   */
  public String getExpectedImplementation();
}
```

The `ExoCacheCreator` allows you to define any kind of jboss cache instance that you would like to have. It has been designed to give you the ability to have your own type of configuration and to always be backward compatible.

In an `ExoCacheCreator`, you need to implement 3 methods which are:

- *create*: this method is used to create a new `ExoCache` from the `ExoCacheConfig` and a jboss cache instance.
- *getExpectedConfigType*: this method is used to indicate the factory the subtype of `ExoCacheConfig` supported by the creator.
- *getExpectedImplementation*: this method is used to indicate the factory and the value of field implementation of `ExoCacheConfig` that is supported by the creator. This is used for backward compatibility, in other words, you can still configure your cache with a super class `ExoCacheConfig`.

Registering a cache creator

You can register any cache creator that you want thanks to an *"external plugin"*, see an example below:

```
<external-component-plugins>
  <target-component>org.exoplatform.services.cache.ExoCacheFactory</target-component>
  <component-plugin>
    <name>addCreator</name>
    <set-method>addCreator</set-method>
    <type>org.exoplatform.services.cache.impl.jboss.ExoCacheCreatorPlugin</type>
    <description>add Exo Cache Creator</description>
    <init-params>
      <object-param>
        <name>LRU</name>
        <description>The lru cache creator</description>
        <object type="org.exoplatform.services.cache.impl.jboss.lru.LRUExoCacheCreator">
          <field name="defaultTimeToLive"><long>1500</long></field>
          <field name="defaultMaxAge"><long>2000</long></field>
        </object>
      </object-param>
    </init-params>
  </component-plugin>
```

```
  </external-component-plugins>
```

In the example above, I call the method *addCreator(ExoCacheCreatorPlugin plugin)* on the current implementation of `ExoCacheFactory` which is actually the jboss cache implementation.

In the *init-params* block, you can define a set of *object-param* blocks and for each *object-param*, we expect any object definition of type `ExoCacheCreator`.

In this example, we register the action creator related to the eviction policy *LRU*.

The cache creators available

By default, no cache creator are defined, so you need to define them yourself by adding them in your configuration files.

LRU Cache Creator - Least Recently Used

```
..
<object-param>
  <name>LRU</name>
  <description>The lru cache creator</description>
  <object type="org.exoplatform.services.cache.impl.jboss.lru.LRUExoCacheCreator">
    <field name="defaultTimeToLive"><long>${my-value}</long></field>
    <field name="defaultMaxAge"><long>${my-value}</long></field>
  </object>
</object-param>
...
```

Table 60.3. Fields description

defaultTimeToLive	This is the default value of the field *timeToLive* described in the section dedicated to this cache type. This value is only used when we define a cache of this type with the old configuration.
defaultMaxAge	This is the default value of the field *maxAge* described in the section dedicated to this cache type. This value is only used when we define a cache of this type with the old configuration.

FIFO Cache Creator - First In, First Out

```
...
<object-param>
```

```
 <name>FIFO</name>
 <description>The fifo cache creator</description>
 <object type="org.exoplatform.services.cache.impl.jboss.fifo.FIFOExoCacheCreator"></object>
</object-param>
...
```

MRU Cache Creator - Most Recently Used

```
...
<object-param>
  <name>MRU</name>
  <description>The mru cache creator</description>
        <object    type="org.exoplatform.services.cache.impl.jboss.mru.MRUExoCacheCreator"></
object>
</object-param>
...
```

LFU Cache Creator - Least Frequently Used

```
...
<object-param>
  <name>LFU</name>
  <description>The lfu cache creator</description>
  <object type="org.exoplatform.services.cache.impl.jboss.lfu.LFUExoCacheCreator">
    <field name="defaultMinNodes"><int>${my-value}</int></field>
  </object>
</object-param>
...
```

Table 60.4. Fields description

defaultMinNodes	This is the default value of the field *minNodes* described in the section dedicated to this cache type. This value is only used when we define a cache of this type with the old configuration.

EA Cache Creator - Expiration Algorithm

```
...
<object-param>
```

```
  <name>EA</name>
  <description>The ea cache creator</description>
  <object type="org.exoplatform.services.cache.impl.jboss.ea.EAExoCacheCreator">
    <field name="defaultExpirationTimeout"><long>2000</long></field>
  </object>
</object-param>
...
```

Table 60.5. Fields description

defaultExpirationTimeout	This is the default value of the field *minNodes* described in the section dedicated to this cache type. This value is only used when we define a cache of this type with the old configuration.

Defining a cache

How to define a cache?

You have 2 ways to define a cache which are:

- At `CacheService` initialization
- With an *"external plugin"*

At `CacheService` initialization

```
...
  <component>
    <key>org.exoplatform.services.cache.CacheService</key>
    <type>org.exoplatform.services.cache.impl.CacheServiceImpl</type>
    <init-params>
      ...
      <object-param>
        <name>fifocache</name>
        <description>The default cache configuration</description>
        <object type="org.exoplatform.services.cache.ExoCacheConfig">
          <field name="name"><string>fifocache</string></field>
          <field name="maxSize"><int>${my-value}</int></field>
          <field name="liveTime"><long>${my-value}</long></field>
          <field name="distributed"><boolean>false</boolean></field>
           <field name="implementation"><string>org.exoplatform.services.cache.FIFOExoCache</
string></field>
```

```
      </object>
    </object-param>
    ...
  </init-params>
 </component>
...
```

In this example, we define a new cache called *fifocache*.

With an *"external plugin"*

```
...
 <external-component-plugins>
  <target-component>org.exoplatform.services.cache.CacheService</target-component>
  <component-plugin>
    <name>addExoCacheConfig</name>
    <set-method>addExoCacheConfig</set-method>
    <type>org.exoplatform.services.cache.ExoCacheConfigPlugin</type>
    <description>add ExoCache configuration component plugin </description>
    <init-params>
      ...
      <object-param>
        <name>fifoCache</name>
        <description>The fifo cache configuration</description>
        <object type="org.exoplatform.services.cache.ExoCacheConfig">
          <field name="name"><string>fifocache</string></field>
          <field name="maxSize"><int>${my-value}</int></field>
          <field name="liveTime"><long>${my-value}</long></field>
          <field name="distributed"><boolean>false</boolean></field>
          <field name="implementation"><string>org.exoplatform.services.cache.FIFOExoCache</
string></field>
        </object>
      </object-param>
...
    </init-params>
  </component-plugin>
 </external-component-plugins>
...
```

In this example, we define a new cache called *fifocache* which is in fact the same cache as in previous example but defined in a different manner.

How to define a distributed or a local cache?

Actually, if you use a custom configuration for your cache as described in a previous section, we will use the cache mode definde in your configuration file.

In case, you decide to use the default configuration template, we use the field *distributed* of your `ExoCacheConfig` to decide. In other words, if the value of this field is false (the default value), the cache will be a local cache, otherwise it will be the cache mode defined in your default configuration template that should be distributed.

How to share a JBoss Cache instance between multiple eXo Cache instances

In order to avoid creating several JBoss Cache instances that consume resources, it is possible to share the same JBoss Cache instance between multiple eXo Cache instances that rely on the same JBoss Cache config. Each eXo Cache instances will then have their own cache region with their own eviction configuration. To allow sharing JBoss Cache instances, you can set the global value at *ExoCacheFactory* level and if needed set the local value at *ExoCacheConfig* level knowing that local value will redefine the global value. Each new *ExoCacheConfig* described below are a sub class of *AbstractExoCacheConfig* that gives access to the parameter *allowShareableCache*, if this parameter is set, it will be the value used otherwise it will use the global value. For all the old *ExoCacheConfig*, only the global value will be used.

LRU Cache - Least Recently Used

- New configuration

```
...
      <object-param>
       <name>lru</name>
       <description>The lru cache configuration</description>
       <object type="org.exoplatform.services.cache.impl.jboss.lru.LRUExoCacheConfig">
         <field name="name"><string>lru</string></field>
         <field name="maxNodes"><int>${my-value}</int></field>
         <field name="minTimeToLive"><long>${my-value}</long></field>
         <field name="maxAge"><long>${my-value}</long></field>
         <field name="timeToLive"><long>${my-value}</long></field>
       </object>
     </object-param>
...
```

Table 60.6. Fields description

maxNodes	

	This is the maximum number of nodes allowed in this region. 0 denotes immediate expiry, -1 denotes no limit.
minTimeToLive	The minimum amount of time (in milliseconds) that a node must be allowed to live after being accessed before it is allowed to be considered for eviction. 0 denotes that this feature is disabled, which is the default value.
maxAge	Lifespan of a node (in milliseconds) regardless of idle time before the node is swept away. 0 denotes immediate expiry, -1 denotes no limit.
timeToLive	The amount of time that a node is not written to or read (in milliseconds) before the node is swept away. 0 denotes immediate expiry, -1 denotes no limit.

- Old configuration

```
...
    <object-param>
      <name>lru-with-old-config</name>
      <description>The lru cache configuration</description>
      <object type="org.exoplatform.services.cache.ExoCacheConfig">
        <field name="name"><string>lru-with-old-config</string></field>
        <field name="maxSize"><int>${my-value}</int></field>
        <field name="liveTime"><long>${my-value}</long></field>
        <field name="implementation"><string>LRU</string></field>
      </object>
    </object-param>
...
```

Table 60.7. Fields description

maxSize	This is the maximum number of nodes allowed in this region. 0 denotes immediate expiry, -1 denotes no limit.
liveTime	The minimum amount of time (in seconds) that a node must be allowed to live after being accessed before it is allowed to be considered for eviction. 0 denotes that this feature is disabled, which is the default value.

Note

For the fields *maxAge* and *timeToLive* needed by JBoss cache, we will use the default values provided by the creator.

FIFO Cache - First In, First Out

- New configuration

```
...
      <object-param>
       <name>fifo</name>
       <description>The fifo cache configuration</description>
       <object type="org.exoplatform.services.cache.impl.jboss.fifo.FIFOExoCacheConfig">
         <field name="name"><string>fifo</string></field>
         <field name="maxNodes"><int>${my-value}</int></field>
         <field name="minTimeToLive"><long>${my-value}</long></field>
       </object>
     </object-param>
...
```

Table 60.8. Fields description

maxNodes	This is the maximum number of nodes allowed in this region. 0 denotes immediate expiry, -1 denotes no limit.
minTimeToLive	The minimum amount of time (in milliseconds) that a node must be allowed to live after being accessed before it is allowed to be considered for eviction. 0 denotes that this feature is disabled, which is the default value.

- Old configuration

```
...
      <object-param>
       <name>fifo-with-old-config</name>
       <description>The fifo cache configuration</description>
       <object type="org.exoplatform.services.cache.ExoCacheConfig">
         <field name="name"><string>fifo-with-old-config</string></field>
         <field name="maxSize"><int>${my-value}</int></field>
         <field name="liveTime"><long>${my-value}</long></field>
```

```
        <field name="implementation"><string>FIFO</string></field>
       </object>
     </object-param>
...
```

Table 60.9. Fields description

maxSize	This is the maximum number of nodes allowed in this region. 0 denotes immediate expiry, -1 denotes no limit.
liveTime	The minimum amount of time (in seconds) that a node must be allowed to live after being accessed before it is allowed to be considered for eviction. 0 denotes that this feature is disabled, which is the default value.

MRU Cache - Most Recently Used

- New configuration

```
...
      <object-param>
       <name>mru</name>
       <description>The mru cache configuration</description>
       <object type="org.exoplatform.services.cache.impl.jboss.mru.MRUExoCacheConfig">
         <field name="name"><string>mru</string></field>
         <field name="maxNodes"><int>${my-value}</int></field>
         <field name="minTimeToLive"><long>${my-value}</long></field>
       </object>
     </object-param>
...
```

Table 60.10. Fields description

maxNodes	This is the maximum number of nodes allowed in this region. 0 denotes immediate expiry, -1 denotes no limit.
minTimeToLive	The minimum amount of time (in milliseconds) that a node must be allowed to live after being accessed before it is allowed to be considered for eviction. 0 denotes that this feature is disabled, which is the default value.

- Old configuration

```
...
    <object-param>
      <name>mru-with-old-config</name>
      <description>The mru cache configuration</description>
      <object type="org.exoplatform.services.cache.ExoCacheConfig">
        <field name="name"><string>mru-with-old-config</string></field>
        <field name="maxSize"><int>${my-value}</int></field>
        <field name="liveTime"><long>${my-value}</long></field>
        <field name="implementation"><string>MRU</string></field>
      </object>
    </object-param>
...
```

Table 60.11. Fields description

maxSize	This is the maximum number of nodes allowed in this region. 0 denotes immediate expiry, -1 denotes no limit.
liveTime	The minimum amount of time (in seconds) that a node must be allowed to live after being accessed before it is allowed to be considered for eviction. 0 denotes that this feature is disabled, which is the default value.

LFU Cache - Least Frequently Used

- New configuration

```
...
     <object-param>
      <name>lfu</name>
      <description>The lfu cache configuration</description>
      <object type="org.exoplatform.services.cache.impl.jboss.lfu.LFUExoCacheConfig">
        <field name="name"><string>lfu</string></field>
        <field name="maxNodes"><int>${my-value}</int></field>
        <field name="minNodes"><int>${my-value}</int></field>
        <field name="minTimeToLive"><long>${my-value}</long></field>
      </object>
    </object-param>
...
```

Table 60.12. Fields description

maxNodes	This is the maximum number of nodes allowed in this region. 0 denotes immediate expiry, -1 denotes no limit.
minNodes	This is the minimum number of nodes allowed in this region. This value determines what the eviction queue should prune down to per pass. e.g. If minNodes is 10 and the cache grows to 100 nodes, the cache is pruned down to the 10 most frequently used nodes when the eviction timer makes a pass through the eviction algorithm.
minTimeToLive	The minimum amount of time (in milliseconds) that a node must be allowed to live after being accessed before it is allowed to be considered for eviction. 0 denotes that this feature is disabled, which is the default value.

- Old configuration

```
...
    <object-param>
      <name>lfu-with-old-config</name>
      <description>The lfu cache configuration</description>
      <object type="org.exoplatform.services.cache.ExoCacheConfig">
        <field name="name"><string>lfu-with-old-config</string></field>
        <field name="maxSize"><int>${my-value}</int></field>
        <field name="liveTime"><long>${my-value}</long></field>
        <field name="implementation"><string>LFU</string></field>
      </object>
    </object-param>
...
```

Table 60.13. Fields description

maxSize	This is the maximum number of nodes allowed in this region. 0 denotes immediate expiry, -1 denotes no limit.
liveTime	The minimum amount of time (in milliseconds) that a node must be allowed to live after being accessed before it is allowed to be considered

	for eviction. 0 denotes that this feature is disabled, which is the default value.

Note

For the fields *minNodes* and *timeToLive* needed by JBoss cache, we will use the default values provided by the creator.

EA Cache - Expiration Algorithm

- New configuration

```
...
      <object-param>
       <name>ea</name>
       <description>The ea cache configuration</description>
       <object type="org.exoplatform.services.cache.impl.jboss.ea.EAExoCacheConfig">
         <field name="name"><string>ea</string></field>
         <field name="maxNodes"><int>${my-value}</int></field>
         <field name="minTimeToLive"><long>${my-value}</long></field>
         <field name="expirationTimeout"><long>${my-value}</long></field>
       </object>
     </object-param>
...
```

Table 60.14. Fields description

maxNodes	This is the maximum number of nodes allowed in this region. 0 denotes immediate expiry, -1 denotes no limit.
minTimeToLive	The minimum amount of time (in milliseconds) that a node must be allowed to live after being accessed before it is allowed to be considered for eviction. 0 denotes that this feature is disabled, which is the default value.
expirationTimeout	This is the timeout after which the cache entry must be evicted.

- Old configuration

```
...
```

```
      <object-param>
        <name>ea-with-old-config</name>
        <description>The ea cache configuration</description>
        <object type="org.exoplatform.services.cache.ExoCacheConfig">
          <field name="name"><string>lfu-with-old-config</string></field>
          <field name="maxSize"><int>${my-value}</int></field>
          <field name="liveTime"><long>${my-value}</long></field>
          <field name="implementation"><string>EA</string></field>
        </object>
      </object-param>
...
```

Table 60.15. Fields description

maxSize	This is the maximum number of nodes allowed in this region. 0 denotes immediate expiry, -1 denotes no limit.
liveTime	The minimum amount of time (in milliseconds) that a node must be allowed to live after being accessed before it is allowed to be considered for eviction. 0 denotes that this feature is disabled, which is the default value.

Note

For the fields *expirationTimeout* needed by JBoss cache, we will use the default values provided by the creator.

eXo Cache based on Infinispan

Configure the ExoCacheFactory

When you add the related jar file in your classpath, the eXo service container will use the default configuration provided in the library itself but of course you can still redefined the configuration if you wish as you can do with any components.

The default configuration of the factory is:

```
<configuration>
  <component>
    <key>org.exoplatform.services.cache.ExoCacheFactory</key>
    <type>org.exoplatform.services.cache.impl.infinispan.ExoCacheFactoryImpl</type>
    <init-params>
      <value-param>
```

```
      <name>cache.config.template</name>
      <value>jar:/conf/portal/cache-configuration-template.xml</value>
    </value-param>
  </init-params>
 </component>
</configuration>
```

As you can see the factory requires one single parameter which is *cache.config.template*, this parameter allows you to define the location of the default configuration template of your infinispan. In the default configuration, we ask the eXo container to get the file shipped into the jar at */conf/portal/cache-configuration-template.xml*.

The default configuration template aims to be the skeleton from which we will create any type of infinispan cache instance, thus it must be very generic.

Note

All the cache instances that will rely on this cache configuration will share the same `EmbeddedCacheManager`.

Add specific configuration for a cache

If for a given reason, you need to use a specific configuration for a cache, you can register one thanks to an "*external plugin*", see an example below:

```
<configuration>
 ...
 <external-component-plugins>
  <target-component>org.exoplatform.services.cache.ExoCacheFactory</target-component>
  <component-plugin>
   <name>addConfig</name>
   <set-method>addConfig</set-method>
   <type>org.exoplatform.services.cache.impl.infinispan.ExoCacheFactoryConfigPlugin</type>
   <description>add Custom Configurations</description>
   <init-params>
    <value-param>
     <name>myCustomCache</name>
     <value>jar:/conf/portal/custom-cache-configuration.xml</value>
    </value-param>
   </init-params>
  </component-plugin>
 </external-component-plugins>
 ...
</configuration>
```

In the example above, I call the method *addConfig(ExoCacheFactoryConfigPlugin plugin)* on the current implementation of `ExoCacheFactory` which is actually the infinispan implementation.

In the *init-params* block, you can define a set of *value-param* blocks and for each *value-param*, we expect the name of cache that needs a specific configuration as name and the location of your custom configuration as *value*.

In this example, we indicates to the factory that we would like that the cache *myCustomCache* use the configuration available at *jar:/conf/portal/custom-cache-configuration.xml*.

Note

All the cache instances that will rely on the cache configuration located at the same location will share the same `EmbeddedCacheManager`.

Add a cache creator

Understanding a cache creator

The factory for infinispan, delegates the cache creation to `ExoCacheCreator` that is defined as below:

```
package org.exoplatform.services.cache.impl.infinispan;
...
public interface ExoCacheCreator {

  /**
          * Creates an eXo cache according to the given configuration {@link
 org.exoplatform.services.cache.ExoCacheConfig}
   * @param config the configuration of the cache to apply
   * @param cacheConfig the configuration of the infinispan cache
   * @param cacheGetter a {@link Callable} instance from which we can get the cache
   * @exception ExoCacheInitException if an exception happens while initializing the cache
   */
          public ExoCache<Serializable, Object> create(ExoCacheConfig config,
  Configuration cacheConfig, Callable<Cache<Serializable, Object>> cacheGetter) throws
 ExoCacheInitException;

  /**
   * Returns the type of {@link org.exoplatform.services.cache.ExoCacheConfig} expected by the
 creator
   * @return the expected type
   */
  public Class<? extends ExoCacheConfig> getExpectedConfigType();
```

```
  /**
   * Returns a set of all the implementations expected by the creator. This is mainly used to be
 backward compatible
   * @return the expected by the creator
   */
  public Set<String> getExpectedImplementations();
}
```

The `ExoCacheCreator` allows you to define any kind of infinispan cache instance that you would like to have. It has been designed to give you the ability to have your own type of configuration and to always be backward compatible.

In an `ExoCacheCreator`, you need to implement 3 methods which are:

- *create* - this method is used to create a new `ExoCache` from the `ExoCacheConfig`, an inifinispan cache configuration and a Callable object to allow you to get the cache instance.
- *getExpectedConfigType* - this method is used to indicate the factory the subtype of `ExoCacheConfig` supported by the creator.
- *getExpectedImplementation*s - this method is used to indicate the factory the values of the field *implementation* of `ExoCacheConfig` that is supported by the creator. This is used for backward compatibility, in other words you can still configure your cache with an instance of `ExoCacheConfig`.

Register a cache creator

You can register any cache creator you want thanks to an *"external plugin"*, see an example below:

```
 <external-component-plugins>
   <target-component>org.exoplatform.services.cache.ExoCacheFactory</target-component>
   <component-plugin>
     <name>addCreator</name>
     <set-method>addCreator</set-method>
     <type>org.exoplatform.services.cache.impl.infinispan.ExoCacheCreatorPlugin</type>
     <description>add Exo Cache Creator</description>
     <init-params>
       <object-param>
         <name>Test</name>
         <description>The cache creator for testing purpose</description>
           <object type="org.exoplatform.services.cache.impl.infinispan.TestExoCacheCreator"></
object>
       </object-param>
     </init-params>
   </component-plugin>
```

```
</external-component-plugins>
```

In the example above, I call the method *addCreator(ExoCacheCreatorPlugin plugin)* on the current implementation of `ExoCacheFactory` which is actually the infinispan implementation.

In the *init-params* block, you can define a set of *object-param* blocks and for each *object-param*, we expect any object definition of type `ExoCacheCreator`.

In this example, we register the cache creator related to the eviction policy *Test*.

The cache creators available

By default, no cache creator are defined, so you need to define them yourself by adding them in your configuration files.

Generic Cache Creator

This is the generic cache creator that allows you to use any eviction strategies defined by default in Infinispan.

```
..
<object-param>
  <name>GENERIC</name>
  <description>The generic cache creator</description>
                                                                                <object
 type="org.exoplatform.services.cache.impl.infinispan.generic.GenericExoCacheCreator">
    <field name="implementations">
      <collection type="java.util.HashSet">
        <value>
          <string>NONE</string>
        </value>
        <value>
          <string>FIFO</string>
        </value>
        <value>
          <string>LRU</string>
        </value>
        <value>
          <string>UNORDERED</string>
        </value>
        <value>
          <string>LIRS</string>
        </value>
      </collection>
    </field>
```

```
    <field name="defaultStrategy"><string>${my-value}</string></field>
    <field name="defaultMaxIdle"><long>${my-value}</long></field>
    <field name="defaultWakeUpInterval"><long>${my-value}</long></field>
  </object>
</object-param>
...
```

Table 60.16. Fields description

implementations	This is the list of all the *implementations* supported by the cache creator. Actualy, it is a subset of the full list of the eviction strategies supported by infinispan to which you want to give access to. In the configuraton above, you have the full list of all the eviction strategies currently supported by infinispan 4.1. This field is used to manage the backward compatibility.
defaultStrategy	This is the name of the default eviction strategy to use. By default the value is *LRU*. This value is only use when we define a cache of this type with the old configuration.
defaultMaxIdle	This is the default value of the field *maxIdle* described in the section dedicated to this cache type. By default the value is *-1*.This value is only use when we define a cache of this type with the old configuration.
defaultWakeUpInterval	his is the default value of the field *wakeUpInterval* described in the section dedicated to this cache type. By default the value is *5000*.This value is only use when we define a cache of this type with the old configuration

Define an infinispan cache instance

How to define a distributed or a local cache?

Actually, if you use a custom configuration for your cache as described in a previous section, we will use the cache mode define in your configuration file.

In case, you decide to use the default configuration template, we use the field *distributed* of your `ExoCacheConfig` to decide. In other words, if the value of this field is false (the default value), the cache will be a local cache otherwise it will be the cache mode defined in your default configuration template that should be distributed.

How to define an infinispan cache instance

All the eviction strategies proposed by default in infinispan rely on the generic cache creator.

- New configuration

```
...
      <object-param>
       <name>myCache</name>
       <description>My cache configuration</description>
                                                                                <object
 type="org.exoplatform.services.cache.impl.infinispan.generic.GenericExoCacheConfig">
         <field name="name"><string>myCacheName</string></field>
         <field name="strategy"><int>${my-value}</int></field>
         <field name="maxEntries"><long>${my-value}</long></field>
         <field name="lifespan"><long>${my-value}</long></field>
         <field name="maxIdle"><long>${my-value}</long></field>
         <field name="wakeUpInterval"><long>${my-value}</long></field>
       </object>
     </object-param>
...
```

Table 60.17. Fields description

strategy	The name of the strategy to use such as 'UNORDERED', 'FIFO', 'LRU', 'LIRS' and 'NONE' (to disable eviction).
maxEntries	Maximum number of entries in a cache instance. If selected value is not a power of two the actual value will default to the least power of two larger than selected value. -1 means no limit which is also the default value.
lifespan	Maximum lifespan of a cache entry, after which the entry is expired cluster-wide, in milliseconds. -1 means the entries never expire which is also the default value.
maxIdle	Maximum idle time a cache entry will be maintained in the cache, in milliseconds. If the idle time is exceeded, the entry will be expired cluster-wide. -1 means the entries never expire which is also the default value.
wakeUpInterval	Interval between subsequent eviction runs, in milliseconds. If you wish to disable the periodic

	eviction process altogether, set wakeupInterval to -1. The default value is 5000.

- Old configuration

```
...
    <object-param>
      <name>myCache</name>
      <description>My cache configuration</description>
        <field name="name"><string>lru-with-old-config</string></field>
        <field name="maxSize"><int>${my-value}</int></field>
        <field name="liveTime"><long>${my-value}</long></field>
        <field name="implementation"><string>${my-value}</string></field>
      </object>
    </object-param>
...
```

Table 60.18. Fields description

maxSize	Maximum number of entries in a cache instance. If selected value is not a power of two the actual value will default to the least power of two larger than selected value. -1 means no limit which is also the default value.
liveTime	Maximum lifespan of a cache entry, after which the entry is expired cluster-wide, in milliseconds. -1 means the entries never expire which is also the default value.
implementation	The name of the implementation to use the expected value is one of the eviction strategies defined in the field *implementations* of the generic cache creator.

Note

For the fields *maxIdle* and *wakeUpInterval* needed by infinispan, we will use the default values provided by the creator.

TransactionService

Base information

TransactionServices provides access to the TransactionManager and the UserTransaction (See JTA specification for details).

Table 61.1. List methods

getTransactionManager()	Get used TransactionManager
getUserTransaction()	Get UserTransaction on TransactionManager
getDefaultTimeout()	Return default TimeOut
setTransactionTimeout(int seconds)	Set TimeOut in second
enlistResource(XAResource xares)	Enlist XA resource in TransactionManager
delistResource(XAResource xares)	Delist XA resource from TransactionManager

Existing TransactionService implementations

eXo JCR proposes out of the box several implementations, they all implement the abstract class *org.exoplatform.services.transaction.impl.AbstractTransactionService*. This main class implement the biggest part of all the methods proposed by the *TransactionService*. For each sub-class of *AbstractTransactionService*, you can set the transaction timeout by configuration using the value parameter *timeout* that is expressed in seconds.

JOTM in standalone mode

To use JOTM as TransactionManager in standalone mode, simply add the following component configuration:

```
  <component>
    <key>org.exoplatform.services.transaction.TransactionService</key>
    <type>org.exoplatform.services.transaction.impl.jotm.TransactionServiceJotmImpl</type>
   <!-- Uncomment the lines below if you want to set default transaction timeout that is expressed
 in seconds -->
    <!--init-params>
      <value-param>
        <name>timeout</name>
        <value>60</value>
      </value-param>
    </init-params-->
  </component>
```

Generic TransactionService based on the TransactionManagerLookup of JBoss Cache

If you intend to use JBoss Cache, you can use a generic TransactionService based on its TransactionManagerLookup which is able to automatically find the TransactionManager of several Application Servers thanks to a set of JNDI lookups. This generic TransactionService covers mainly the TransactionManager lookups, the UserTransaction is actually simply the TransactionManager instance that has been wrapped. See below an example of configuration:

```
  <!-- Configuration of the TransactionManagerLookup -->
  <component>
    <key>org.jboss.cache.transaction.TransactionManagerLookup</key>
    <type>org.jboss.cache.transaction.GenericTransactionManagerLookup</type>
  </component>
  <!-- Configuration of the TransactionService -->
  <component>
    <key>org.exoplatform.services.transaction.TransactionService</key>
    <type>org.exoplatform.services.transaction.jbosscache.GenericTransactionService</type>
   <!-- Uncomment the lines below if you want to set default transaction timeout that is expressed
 in seconds -->
    <!--init-params>
      <value-param>
        <name>timeout</name>
        <value>60</value>
      </value-param>
    </init-params-->
  </component>
```

Specific GenericTransactionService for JBoss Cache and Arjuna

If you intend to use JBoss Cache with Arjuna, you can use a more specific GenericTransactionService, it is mostly interesting in case you want to use the real UserTransaction. See below an example of configuration:

```
  <!-- Configuration of the TransactionManagerLookup -->
  <component>
    <key>org.jboss.cache.transaction.TransactionManagerLookup</key>
    <type>org.jboss.cache.transaction.JBossStandaloneJTAManagerLookup</type>
  </component>
  <!-- Configuration of the TransactionService -->
  <component>
```

```
    <key>org.exoplatform.services.transaction.TransactionService</key>
    <type>org.exoplatform.services.transaction.jbosscache.JBossTransactionsService</type>
   <!-- Uncomment the lines below if you want to set default transaction timeout that is expressed
in seconds -->
    <!--init-params>
      <value-param>
        <name>timeout</name>
        <value>60</value>
      </value-param>
    </init-params-->
  </component>
```

Generic TransactionService based on the TransactionManagerLookup of Infinispan

If you intend to use Infinispan, you can use a generic TransactionService based on its TransactionManagerLookup which is able to automatically find the TransactionManager of several Application Servers thanks to a set of JNDI lookups. This generic TransactionService covers mainly the TransactionManager lookups, the UserTransaction is actually simply the TransactionManager instance that has been wrapped. See below an example of configuration:

```
  <!-- Configuration of the TransactionManagerLookup -->
  <component>
    <key>org.infinispan.transaction.lookup.TransactionManagerLookup</key>
    <type>org.infinispan.transaction.lookup.GenericTransactionManagerLookup</type>
  </component>
  <!-- Configuration of the TransactionService -->
  <component>
    <key>org.exoplatform.services.transaction.TransactionService</key>
    <type>org.exoplatform.services.transaction.infinispan.GenericTransactionService</type>
   <!-- Uncomment the lines below if you want to set default transaction timeout that is expressed
in seconds -->
    <!--init-params>
      <value-param>
        <name>timeout</name>
        <value>60</value>
      </value-param>
    </init-params-->
  </component>
```

Specific GenericTransactionService for Infinispan and Arjuna

If you intend to use Infinispan with Arjuna, you can use a more specific GenericTransactionService, it is mostly interesting in case you want to use the real UserTransaction. See below an example of configuration:

```
  <!-- Configuration of the TransactionManagerLookup -->
  <component>
    <key>org.infinispan.transaction.lookup.TransactionManagerLookup</key>


type>
  </component>
  <!-- Configuration of the TransactionService -->
  <component>
    <key>org.exoplatform.services.transaction.TransactionService</key>
    <type>org.exoplatform.services.transaction.infinispan.JBossTransactionsService</type>
   <!-- Uncomment the lines below if you want to set default transaction timeout that is expressed
 in seconds -->
    <!--init-params>
      <value-param>
        <name>timeout</name>
        <value>60</value>
      </value-param>
    </init-params-->
  </component>
```

A very specific TransactionService for JBoss AS

If you intend to use JBoss AS with JBoss Cache and Infinispan, you can use a very specific TransactionService for JBoss AS. See below an example of configuration:

```
  <component>
    <key>org.exoplatform.services.transaction.TransactionService</key>
    <type>org.exoplatform.services.transaction.impl.jboss.JBossTransactionService</type>
   <!-- Uncomment the lines below if you want to set default transaction timeout that is expressed
 in seconds -->
    <!--init-params>
      <value-param>
        <name>timeout</name>
        <value>60</value>
      </value-param>
```

```
    </init-params-->
  </component>
```

TransactionsEssentials in standalone mode

To use TransactionsEssentials as TransactionManager in standalone mode, simply add the following component configuration:

```
  <component>
    <key>org.exoplatform.services.transaction.TransactionService</key>


type>
    <!-- Uncomment the lines below if you want to set default transaction timeout that is expressed
 in seconds -->
    <!--init-params>
      <value-param>
        <name>timeout</name>
        <value>60</value>
      </value-param>
    </init-params-->
  </component>
```

The data source provider

Description

The *DataSourceProvider* is a service used to give access to a data source in an uniform manner in order to be able to support data sources that are managed by the application server.

Table 62.1. List methods

getDataSource(String dataSourceName)	Tries to get the data source from a JNDI lookup. If it can be found and the data source is defined as managed, the service will wrap the original *DataSource* instance in a new *DataSource* instance that is aware of its *managed* state otherwise it will return the original *DataSource* instance.
isManaged(String dataSourceName)	Indicates whether or not the given data source is managed.

Configuration

The configuration of the *DataSourceProvider* should be defined only if you use managed data sources since by default all the data sources are considered as not managed. See below the default configuration

```
<configuration>
....
  <component>
    <key>org.exoplatform.services.jdbc.DataSourceProvider</key>
    <type>org.exoplatform.services.jdbc.impl.DataSourceProviderImpl</type>
    <init-params>
      <!--  Indicates that the data source needs to check if a tx is active
          to decide if the provided connection needs to be managed or not.
          If it is set to false, the data source will provide only
          managed connections if the data source itself is managed.  -->
      <!--value-param>
        <name>check-tx-active</name>
        <value>true</value>
      </value-param-->
      <!-- Indicates that all the data sources are managed
          If set to true the parameter never-managed and
          managed-data-sources will be ignored -->
      <!--value-param>
```

```
          <name>always-managed</name>
          <value>true</value>
        </value-param-->
        <!-- Indicates the list of all the data sources that are
             managed, each value tag can contain a list of
             data source names separated by a comma, in the
             example below we will register ds-foo1, ds-foo2
             and ds-foo3 as managed data source. If always-managed
             and/or never-managed is set true this parameter is ignored -->
        <!--values-param>
          <name>managed-data-sources</name>
          <value>ds-foo1, ds-foo2</value>
          <value>ds-foo3</value>
        </values-param-->
      </init-params>
    </component>
...
</configuration>
```

Table 62.2. Fields description

check-tx-active	This parameter indicates that the data source needs to check if a transaction is active to decide if the provided connection needs to be managed or not. If it is set to *false*, the data source will provide only managed connections if the data source itself is managed. By default, this parameter is set to *true*. If this parameter is set to *true*, it will need the *TransactionService* to work propertly, so please ensure that the *TransactionService* is defined in your configuration.
always-managed	This parameter indicates that all the data sources are managed. If set to *true* the parameter *never-managed* and *managed-data-sources* will be ignored, so it will consider all the data sources as managed. By default, this parameter is set to *false*.
managed-data-sources	This parameter indicates the list of all the data sources that are managed, each value tag can contain a list of data source names separated by a comma. If *always-managed* and/or *never-managed* is set *true* this parameter is ignored.

JNDI naming

Prerequisites

We need to configure JNDI environment properties and Reference binding with the eXo container standard mechanism.

The Naming service covers:

- Configuring the current Naming Context Factory implemented as an ExoContainer Component `org.exoplatform.services.naming.InitialContextInitializer`.
- Binding Objects (References) to the current Context using `org.exoplatform.services.naming.BindReferencePlugin` component plugin.

How it works

Make sure you understand the *Java Naming and Directory InterfaceTM (JNDI)* [http://java.sun.com/products/jndi/1.2/javadoc/index.html] concepts before using this service.

JNDI System property initialization

After the start time the Context Initializer (org.exoplatform.services.naming.InitialContextInitializer) traverses all initial parameters (that concern the Naming Context) configured in `default-properties` and `mandatory-properties` (see Configuration examples) and:

- For `default-properties`: Check if this property is already set as a System property (`System.getProperty(name)`) and set this property if it's not found. Using those properties is recommended with a third party Naming service provider.
- For `mandatory-properties`: Set the property without checking.

Standard JNDI properties:

- `java.naming.factory.initial`
- `java.naming.provider.url`

and others (see JNDI docs)

JNDI reference binding

Another responsibility of Context Initializer `org.exoplatform.services.naming.InitialContextInitializer` is binding of preconfigured

references to the naming context. For this purpose, it uses a standard eXo component plugin mechanism and in particular the `org.exoplatform.services.naming.BindReferencePlugin` component plugin. The configuration of this plugin includes three mandatory value parameters:

- `bind-name`: the name of binding reference.
- `class-name`: the type of binding reference.
- `factory`: the object factory type.

And also `ref-addresses` property parameter with a set of references' properties. (see Configuration examples) Context Initializer uses those parameters to bind the neccessary reference automatically.

Configuration examples

The `InitialContextInitializer` configuration example:

```
<component>
  <type>org.exoplatform.services.naming.InitialContextInitializer</type>
  <init-params>
    <value-param>.
      <name>bindings-store-path</name>.
      <value>bind-references.xml</value>.
    </value-param>.
    <value-param>
      <name>overload-context-factory</name>
      <value>true</value>
    </value-param>
    <properties-param>
      <name>default-properties</name>
      <description>Default initial context properties</description>
                                                <property name="java.naming.factory.initial"
value="org.exoplatform.services.naming.SimpleContextFactory"/>
    </properties-param>
    <properties-param>
      <name>mandatory-properties</name>
      <description>Mandatory initial context properties</description>
      <property name="java.naming.provider.url" value="rmi://localhost:9999"/>
    </properties-param>
  </init-params>
</component>
```

where

binding-store-path is file path which stores binded datasources in runtime

overload-context-factory allows to overload the default initial context factory by a context factory that is ExoContainer aware and that is able to delegate to the original initial context factory if it detects that it is not in the eXo scope. By default the feature is disabled since it is only required on AS that don't share the objects by default like tomcat but unlike JBoss AS

The `BindReferencePlugin` component plugin configuration example (for JDBC datasource):

```
<component-plugins>
  <component-plugin>
    <name>bind.datasource</name>
    <set-method>addPlugin</set-method>
    <type>org.exoplatform.services.naming.BindReferencePlugin</type>
    <init-params>
      <value-param>
        <name>bind-name</name>
        <value>jdbcjcr</value>
      </value-param>
      <value-param>
        <name>class-name</name>
        <value>javax.sql.DataSource</value>
      </value-param>
      <value-param>
        <name>factory</name>
        <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
      </value-param>
      <properties-param>
        <name>ref-addresses</name>
        <description>ref-addresses</description>
        <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
        <property name="url" value="jdbc:hsqldb:file:target/temp/data/portal"/>
        <property name="username" value="sa"/>
        <property name="password" value=""/>
      </properties-param>
    </init-params>
  </component-plugin>
```

Recommendations for Application Developers

- `SimpleContextFactory` is created for testing purposes only, do not use it for production.
- In J2EE environment use Naming Factory objects provided with the Application Server.

InitialContextInitializer API

`InitialContextInitalizer` also provides feature of references binding in runtime. References have bind in runtime will be persisted and automatically rebinded on a next system start.

Service provides methods for binding reference.

```
public void bind(String bindName,
                 String className,
                 String factory,
                 String factoryLocation,
                 Map<String, String> refAddr)
          throws NamingException, FileNotFoundException, XMLStreamException;
```

- `bindName`: name of binding.
- `className`: the fully-qualified name of the class of the object to which this Reference refers.
- `factory`: the name of the factory class for creating an instance of the object to which this Reference refers.
- `factoryLocation`: the location of the factory class.
- `refAddr`: object's properties map.

Example of usage:

```
// obtain InitialContextInitializer instance from ExoContainer (e.g. PortalContainer)
InitialContextInitializer initContext = (InitialContextInitializer)container.getComponentInstanceOfType(InitialContextInitializer.class);

Map<String, String> refAddr = new HashMap<String, String>();
refAddr.put("driverClassName", "oracle.jdbc.OracleDriver");
refAddr.put("url", "jdbc:oracle:thin:@oraclehost:1521:orcl");
refAddr.put("username", "exouser");
refAddr.put("password", "exopassword");

initContext.bind("jdbcexco", "javax.sql.DataSource", "org.apache.commons.dbcp.BasicDataSourceFactory", null, refAddr);

// try to get just bound DataSource
DataSource ds = (DataSource)new InitialContext().lookup("jdbcexo");
```

Logs configuration

Introdution

In order to accommodate to the different target runtime where it can be deployed, eXo is capable of leveraging several logging systems. eXo lets you choose the underlying logging engine to use and even configure that engine (as a quick alternative to doing it directly in your runtime environment).

The currently supported logging engines are :

- Apache Log4J
- JDK's logging
- Apache Commons logging (which is itself a pluggable logging abstraction)

Logs configuration initializer

eXo lets you choose whatever logging engine you want as this is generally influences by the AS runtime or internal policy.

This is done through an eXo component called `LogConfigurationInitializer`.

`org.exoplatform.services.log.LogConfigurationInitializer` that reads init parameters and configures logging system according to them. The parameters:

- *configurator* - an implementation of the `LogConfigurator` interface with one method configure() that accepts a list of properties (3rd init parameter) to configure the underlying log system using the concrete mechanism. Again, there are three configurators for the most known log systems (commons, log4j, jdk).
- *properties* - properties to configure the concrete log system (system properties for commons, log4j.properties or logging.properties for commons, log4j and jdk respectively) Look at the configuration examples below.
- *logger* - an implementation of commons-logging Log interface. It is possible to use commons wrappers but to support buffering required by the log portlet three kinds of loggers were added: `BufferedSimpleLog`, `BufferedLog4JLogger` and `BufferedJdk14Logger` (they contain BufferedLog and extend SimpleLog, Log4JLogger and Jdk14Logger commons-logging wrappers respectively).

Configuration examples

Log4J

Log4J [http://logging.apache.org/log4j/] is a very popular and flexible logging system. It is a good option for JBoss.

```
<component>
  <type>org.exoplatform.services.log.LogConfigurationInitializer</type>
  <init-params>
    <value-param>
      <name>logger</name>
      <value>org.exoplatform.services.log.impl.BufferedLog4JLogger</value>
    </value-param>
    <value-param>
      <name>configurator</name>
      <value>org.exoplatform.services.log.impl.Log4JConfigurator</value>
    </value-param>
    <properties-param>
      <name>properties</name>
      <description>Log4J properties</description>
      <property name="log4j.rootLogger" value="DEBUG, stdout, file"/>
      <property name="log4j.appender.stdout" value="org.apache.log4j.ConsoleAppender"/>
      <property name="log4j.appender.stdout.layout" value="org.apache.log4j.PatternLayout"/>
      <property name="log4j.appender.stdout.layout.ConversionPattern" value="%d {dd.MM.yyyy
HH:mm:ss} %c {1}: %m (%F, line %L) %n"/>
      <property name="log4j.appender.file" value="org.apache.log4j.FileAppender"/>
      <property name="log4j.appender.file.File" value="jcr.log"/>
      <property name="log4j.appender.file.layout" value="org.apache.log4j.PatternLayout"/>
          <property name="log4j.appender.file.layout.ConversionPattern" value="%d{dd.MM.yyyy
HH:mm:ss} %m (%F, line %L) %n"/>
    </properties-param >
  </init-params>
 </component>
```

Assigning logger level for classes or components

You can set logger level for class or group of classes by setting next property:

```
<property name="log4j.category.{component or class name}" value="DEBUG"/>
```

For example:

- We want to log all debug messages for class `org.exoplatform.services.jcr.impl.core.SessionDataManager`, that lies in exo.jcr.component.core component

```
<property name="log4j.category.exo.jcr.component.core.SessionDataManager"
 value="DEBUG"/>
```

- Or we want to log all debug messages for all classes in exo.jcr.component.core component

```
<property name="log4j.category.exo.jcr.component.core" value="DEBUG"/>
```

- Or we want to log all messages for all kernel components

```
<property name="log4j.category.exo.kernel" value="DEBUG"/>
```

JDK Logging

JDK logging (aka JUL) is the builtin logging framework introduced in JDK 1.4. It is a good option for Tomcat AS.

- Edit the variable `LOG_OPTS` in your `eXo.sh` or `eXo.bat` :

```
LOG_OPTS="-
Dorg.apache.commons.logging.Log=org.apache.commons.logging.impl.Jdk14Logger"
```

- Edit your `logs-configuration.xml:`

```
<component>
   <type>org.exoplatform.services.log.LogConfigurationInitializer</type>
   <init-params>
     <value-param>
       <name>logger</name>
       <value>org.exoplatform.services.log.impl.BufferedJdk14Logger</value>
     </value-param>
     <value-param>
       <name>configurator</name>
       <value>org.exoplatform.services.log.impl.Jdk14Configurator</value>
     </value-param>
     <properties-param>
       <name>properties</name>
       <description>jdk1.4 Logger properties</description>
       <property name="handlers" value="java.util.logging.ConsoleHandler"/>
```

```
      <property name=".level" value="FINE"/>
      <property name="java.util.logging.ConsoleHandler.level" value="FINE"/>
    </properties-param>
  </init-params>
</component>
```

Commons Logging SimpleLogss

SimpleLog is a minimal logging system distributed with Commons Logging. To be used when nothing else is available or when you seek simplicity.

```
<component>
  <type>org.exoplatform.services.log.LogConfigurationInitializer</type>
  <init-params>
    <value-param>
      <name>logger</name>
      <value>org.exoplatform.services.log.impl.BufferedSimpleLog</value>
    </value-param>
    <value-param>
      <name>configurator</name>
      <value>org.exoplatform.services.log.impl.SimpleLogConfigurator</value>
    </value-param>
    <properties-param>
      <name>properties</name>
      <description>SimpleLog properties</description>
      <property name="org.apache.commons.logging.simplelog.defaultlog" value="debug"/>
      <property name="org.apache.commons.logging.simplelog.showdatetime" value="true"/>
    </properties-param>
  </init-params>
</component>
```

Tips and Troubleshooting

JBoss tips

If you use log4j configuration, you can change the log configuration directly at runtime in: `JBOSS_HOME/server/default/conf/jboss-log4j.xml`.

- To enable debug logs:

```
<param name="Threshold" value="DEBUG"/>
```

- To exclude messages from unnecessary classes (server's internal) modify the threshold of these classes to "FATAL".

> If you see only ERROR level logs while starting ear on jboss (4.2.2), you have to remove log4j*.jar from your ear and application.xml.

Other tips

If you see only ERROR level logs while starting ear on jboss (4.2.2), you have to remove log4j*.jar from your ear and application.xml.

Manageability

Introduction

The kernel has a framework for exposing a management view of the various sub systems of the platform. The management view is a lose term for defining how we can access relevant information about the system and how we can apply management operations. JMX is the de facto standard for exposing a management view in the Java Platform but we take in consideration other kind of views such as REST web services. Therefore, the framework is not tied to JMX, yet it provides a JMX part to define more precisely details related to the JMX management view. The legacy framework is still in use but is deprecated in favor of the new framework as it is less tested and less efficient. It will be removed by sanitization in the future.

Managed framework API

The managed frameworks defines an API for exposing a management view of objects. The API is targeted for internal use and is not a public API. The framework leverages Java 5 annotations to describe the management view from an object.

Annotations

@org.exoplatform.management.annotations.Managed annotation

The @Managed annotates elements that wants to expose a management view to a management layer.

@Managed for objects

The framework will export a management view for the objects annotated.

@Managed for getter/setter

Defines a managed property. An annotated getter defines a read property, an annotated setter defines a write property and if matching getter/setter are annotated it defines a read/write property.

@Managed on method

Defines a managed operation.

@org.exoplatform.management.annotations.ManagedDescription

The @ManagedDescription annotation provides a description of a managed element. It is valid to annotated object or methods. It takes as sole argument a string that is the description value.

@org.exoplatform.management.annotations.ManagedName

The @ManagedName annotation provides an alternative name for managed properties. It is used to accomodate legacy methods of an object that can be renamed for compatibility reasons. It takes as sole argument a string that is the name value.

@org.exoplatform.management.annotations.ManagedBy

The @ManagedBy annotation defines a delegate class for exposing a management view. The sole argument of the annotation are class litterals. The delegate class must provide a constructor with the managed object as argument.

JMX Management View

JMX Annotations

@org.exoplatform.management.jmx.annotations.Property annotation

The @Property annotation is used to within other annotations such as @NameTemplate or @NamingContext. It should be seen as a structural way for a list of properties. A property is made of a key and a value. The value can either be a string litteral or it can be surrounded by curly brace to be a dynamic property. A dynamic property is resolved against the instance of the object at runtime.

@org.exoplatform.management.jmx.annotations.NameTemplate annotation

The @NameTemplate defines a template that is used at registration time of a managed object to create the JMX object name. The template is formed of properties.

```
@NameTemplate({
  @Property(key="container", value="workspace"),
  @Property(key="name", value="{Name}")})
```

@org.exoplatform.management.jmx.annotations.NamingContext annotation

The @NamingContext annotations defines a set of properties which are used within a management context. It allows to propagate properties down to managed objects which are defined by an object implementing the ManagementAware interface. The goal is to scope different instances of the same class that would have the same object name otherwise.

```
@NamingContext(@Property(key="workspace", value="{Name}"))
```

Example

CacheService example

The cache service delegates most of the work to the CacheServiceManaged class by using the @ManagedBy annotation. At runtime when a new cache is created, it calls the CacheServiceManaged class in order to let the CacheServiceManaged object register the cache.

```
@ManagedBy(CacheServiceManaged.class)
public class CacheServiceImpl implements CacheService {

  CacheServiceManaged managed;
  ...
  synchronized private ExoCache createCacheInstance(String region) throws Exception {
    ...
    if (managed != null) {
      managed.registerCache(simple);
    }
    ...
  }
}
```

The ExoCache interface is annotated to define its management view. The @NameTemplate is used to produce object name values when ExoCache instance are registered.

```
@Managed
@NameTemplate({@Property(key="service", value="cache"), @Property(key="name",
 value="{Name}")})
@ManagedDescription("Exo Cache")
public interface ExoCache {

  @Managed
  @ManagedName("Name")
  @ManagedDescription("The cache name")
  public String getName();

  @Managed
  @ManagedName("Capacity")
  @ManagedDescription("The maximum capacity")
```

```
  public int getMaxSize();

  @Managed
  @ManagedDescription("Evict all entries of the cache")
  public void clearCache() throws Exception;

  ...
}
```

The CacheServiceManaged is the glue code between the CacheService and the management view. The main reason is that only exo services are registered automatically against the management view. Any other managed bean must be registered manually for now. Therefore, it needs to know about the management layer via the management context. The management context allows an object implementing the ManagementAware interface to receive a context to perform further registration of managed objects.

```
@Managed
public class CacheServiceManaged implements ManagementAware {

  /** . */
  private ManagementContext context;

  /** . */
  private CacheServiceImpl cacheService;

  public CacheServiceManaged(CacheServiceImpl cacheService) {
    this.cacheService = cacheService;

    //
    cacheService.managed = this;
  }

  public void setContext(ManagementContext context) {
    this.context = context;
  }

  void registerCache(ExoCache cache) {
    if (context != null) {
      context.register(cache);
    }
  }
}
```

ListenerService

Asynchronous Event Broadcast

Basicaly, ListenerService used to store Listeners and broadcast events to them.

ListenerService event broadcasting works in next way - it takes a destination listeners and executes event on those listeners.

But, some events may take a lot of time, so idea to make event processing asynchronous is usefull.

- It's very simple, just mark your Listener implementation as `@Asynchronous`.

```
@Asynchronous
class AsynchListenerWithException<S,D> extends Listener<S,D>
{
  @Override
  public void onEvent(Event<S,D> event) throws Exception
  {
    // some expensive operation
  }
}
```

Now, our AsynchListener will be executed in separate thread by `ExecutorService`.

By default, `ExecutoreService` configured with thread pool size 1, you can change it in configuration:

```
  <component>
    <key>org.exoplatform.services.listener.ListenerService</key>
    <type>org.exoplatform.services.listener.ListenerService</type>

    <init-params>
      <value-param>
        <name>asynchPoolSize</name>
        <value>5</value>
      </value-param>
    </init-params>

  </component>
```

RPC Service

Description

The *RPCService* is only needed in a cluser environment, it is used to communicate with the other cluster nodes. It allows to execute a command on all the cluster nodes or on the coordinator i.e. the oldest node in the cluster. The *RPCService* has been designed to rely on JGroups capabilites and should not be used for heavy load. It can be used for example to notify other nodes that something happened or to collect some information from the other nodes.

The *RPCService* relies on 3 main interfaces which are:

- The *org.exoplatform.services.rpc.RPCService* that defines the service itslef
- The *org.exoplatform.services.rpc.RemoteCommand* that defines the command that we can execute on other nodes.
- The *org.exoplatform.services.rpc.TopologyChangeListener* that defines the listeners that will be notified anytime the topology of the cluster changes.

The arguments that will be given to the *RemoteCommand* must be *Serializable* and its return type also in order to prevent any issue due to the serialization. To prevent to execute any *RemoteCommand* that could be malicious and to allow to use non *Serializable* command, you need to register the command first before using it. Since the service will keep only one instance of *RemoteCommand* per command Id, the implementation of the *RemoteCommand* must be thread safe.

To be usable, all the *RemoteCommand*s must be registered before being used on all the cluster nodes, which means that the command registration must be done in the constructor of your component in other words before that the *RPCService* is started. If you try to launch a command that has been registered but the *RPCService* is not yet launched, you will get an *RPCException* due to an illegal state. This has for consequences that you will be able to execute a command only once your component will be started.

See an example below:

```
public class MyService implements Startable
{
  private RPCService rpcService;
  private RemoteCommand sayHelloCommand;

  public MyService(RPCService rpcService)
  {
    this.rpcService = rpcService;
    // Register the command before that the RPCService is started
    sayHelloCommand = rpcService.registerCommand(new RemoteCommand()
```

```
    {
      public Serializable execute(Serializable[] args) throws Throwable
      {
        System.out.println("Hello !");
        return null;
      }

      public String getId()
      {
        return "hello-world-command";
      }
    });
  }

  public void start()
  {
    // Since the RPCService is a dependency of RPCService, it will be started before
    // so I can execute my command
    try
    {
      // This will make all the nodes say "Hello !"
      rpcService.executeCommandOnAllNodes(sayHelloCommand, false);
    }
    catch (SecurityException e)
    {
      e.printStackTrace();
    }
    catch (RPCException e)
    {
      e.printStackTrace();
    }
  }

  public void stop()
  {
  }
}
```

In the previous example, We register the command *sayHelloCommand* in the constructor of *MyService* and we execute this command in the start method.

Note

We expect to have one *RPCService* instance per *PortalContainer* in a portal mode and only one *RPCService* instance in a standalone mode

Configuration

The configuration of the *RPCService* should be added only in a cluster environment. See below an example of configuration

```
<configuration>
....
  <component>
    <key>org.exoplatform.services.rpc.RPCService</key>
    <type>org.exoplatform.services.rpc.impl.RPCServiceImpl</type>
    <init-params>
      <value-param>
        <name>jgroups-configuration</name>
        <value>classpath:/udp.xml</value>
      </value-param>
      <value-param>
        <name>jgroups-cluster-name</name>
        <value>RPCService-Cluster</value>
      </value-param>
      <value-param>
        <name>jgroups-default-timeout</name>
        <value>0</value>
      </value-param>
      <value-param>
        <name>allow-failover</name>
        <value>true</value>
      </value-param>
      <value-param>
        <name>retry-timeout</name>
        <value>20000</value>
      </value-param>
    </init-params>
  </component>
...
</configuration>
```

Table 67.1. Fields description

jgroups-configuration	

	This is the location of the configuration of jgroups. This parameter is mandatory.
jgroups-cluster-name	This is the name of the cluster. This parameter is optional and its default value is *RPCService-Cluster*. Since we could have several instances of the *RPCService*, the final name will be "${jgroups-cluster-name}-${container-name}"
jgroups-default-timeout	This is the default timeout to use if the timeout is not given, if no response could be get after this timeout an exception will be thrown. This parameter is optional and its default value is 0 which means that we don't use any timeout by default. This parameter is expressed in milliseconds.
allow-failover	This is parameter indicates whether a command on the coordinator needs to be relaunched or not if the coordintator seems to have left the cluster. This parameter only affects the behavior of the methods *executeCommandOnCoordinator*. This parameter is optional and its default value is true.
retry-timeout	This parameter is the maximum amount of time to wait until the new coordinator is elected. This parameter is linked to the parameter *allow-failover*, and thus used in the exact same conditions. This parameter is optional and its default value is 20000. This parameter is expressed in milliseconds.

The SingleMethodCallCommand

Most of the time we only need to call a method on a given object, this can be done thanks to the *org.exoplatform.services.rpc.SingleMethodCallCommand* which is the implementation of a *RemoteCommand* proposed by default. This command will dynamically execute a method on a given object.

```
// Register the command first (to be done before that the RPCService has been started)
RemoteCommand commandGetName = rpcService.registerCommand(new
 SingleMethodCallCommand(myService, "getName"));
...
// Execute the command on the coordinator (can be done only after having started the
 RPCService)
```

```
String name = rpcService.executeCommandOnCoordinator(commandGetName, true);
// Print the name
System.out.println("Name : " + name);
```

This example:

1. Register a *SingleMethodCallCommand* that will call *getName()* on the Object *myService* anytime the command will be executed.
2. Execute the command synchronously on the coordinator, assuming that the same command (with the same id) has already been registered on the coordinator
3. Print the name got from the coordinator

Note

As any *RemoteCommand*, it has to be registered before being executed and before the *RPCService* is launched.

Note

As any *RemoteCommand*, the command can be executed only once the *RPCService* is launched.

Note

The *SingleMethodCallCommand* only allow public methods, if you try to register a non public method an *RPCException* will be thrown at creation level.

Part III. eXoCore

eXo Core

eXo Core introduction

The eXo Core is a set of common services that are used by eXo products and modules, it also can be used in the business logic.

It's Authentication and Security, Organization, Database, Logging, JNDI, LDAP, Document reader and other services.

Database Creator

About

Database creator `DBCreator` is responsible for execution DDL script in runtime. A DDL script may contain templates for database name, user name and password which will be replaced by real values at execution time.

Three templates supported:

- `${database}` for database name;
- `${username}` for user name;
- `${password}` for user's password;

API

Service provide method for execute script for new database creation. Database name which are passed as parameter will be substituted in DDL script instead of `${database}` template. Returns `DBConnectionInfo` object (with all neccesary information of new database's connection) or throws `DBCreatorException` exception if any errors occurs in other case.

```
public DBConnectionInfo createDatabase(String dbName) throws DBCreatorException;
```

For MSSQL and Sybase servers, use autocommit mode to set true for connection. It's due to after execution "create database" command newly created database not available for "use" command and therefore you can't create new user inside database per one script.

```
public DBConnectionInfo getDBConnectionInfo(String dbName) throws DBCreatorException;
```

Return database connection information without database creation.

A configuration examples

Service's configuration.

```
<component>
  <key>org.exoplatform.services.database.creator.DBCreator</key>
  <type>org.exoplatform.services.database.creator.DBCreator</type>
  <init-params>
  <properties-param>
```

```
      <name>db-connection</name>
      <description>database connection properties</description>
      <property name="driverClassName" value="com.mysql.jdbc.Driver" />
      <property name="url" value="jdbc:mysql://localhost/" />
      <property name="username" value="root" />
      <property name="password" value="admin" />
      <property name="additional_property" value="value">
      ...
      <property name="additional_property_n" value="value">
    </properties-param>
    <properties-param>
      <name>db-creation</name>.
      <description>database creation properties</description>.
      <property name="scriptPath" value="script.sql" />
      <property name="username" value="testuser" />
      <property name="password" value="testpwd" />
    </properties-param>
  </init-params>
</component>
```

`db-connection` properties section contains parameters needed for connection to database server

There is four reserved and mandatory properties *driverClassName*, *url*, *username* and *password*. But db-connection may contain additonal properties.

For example, next additional proprites allows reconnect to MySQL database when connection was refused:

```
    <properties-param>
      <name>db-connection</name>
      ...
      <property name="validationQuery" value="select 1"/>
      <property name="testOnReturn" value="true"/>
      ...
    </properties-param>
```

`db-creation` properties section contains paramaters for database creation using DDL script:

- `scriptPath`: absolute path to DDL script file;
- `username`: user name for substitution `${username}` template in DDL script;
- `password`: user's password for substitution `${password}` template in DDL script;

Specific `db-connection` properties section for different databases.

MySQL:

```
<property name="driverClassName" value="com.mysql.jdbc.Driver" />
<property name="url" value="jdbc:mysql://localhost/" />
<property name="username" value="root" />
<property name="password" value="admin" />
```

PostgreSQL:

```
<property name="driverClassName" value="org.postgresql.Driver" />
<property name="url" value="jdbc:postgresql://localhost/" />
<property name="username" value="root" />
<property name="password" value="admin" />
```

MSSQL:

```
<property name="driverClassName" value="com.microsoft.sqlserver.jdbc.SQLServerDriver"/>
<property name="url" value="jdbc:sqlserver://localhost:1433;"/>
<property name="username" value="root"/>
<property name="password" value="admin"/>
```

Sybase:

```
<property name="driverClassName" value="com.sybase.jdbc3.jdbc.SybDriver" />
<property name="url" value="jdbc:sybase:Tds:localhost:5000/"/>
<property name="username" value="root"/>
<property name="password" value="admin"/>
```

Oracle:

```
<property name="driverClassName" value="oracle.jdbc.OracleDriver" />
<property name="url" value="jdbc:oracle:thin:@db2.exoua-int:1521:orclvm" />
<property name="username" value="root" />
<property name="password" value="admin" />
```

An examples of a DDL script

MySQL:

```
CREATE DATABASE ${database};
USE ${database};
CREATE USER '${username}' IDENTIFIED BY '${password}';
GRANT SELECT,INSERT,UPDATE,DELETE ON ${database}.* TO '${username}';
```

PostgreSQL:

```
CREATE USER ${username} WITH PASSWORD '${password}';
CREATE DATABASE ${database} WITH OWNER ${username};
```

MSSQL:

```
USE MASTER;
CREATE DATABASE ${database};
USE ${database};
CREATE LOGIN ${username} WITH PASSWORD = '${password}';
CREATE USER ${username} FOR LOGIN ${username};
```

Sybase:

```
sp_addlogin ${username}, ${password};
CREATE DATABASE ${database};
USE ${database};
sp_adduser ${username};
```

Oracle:

```
CREATE TABLESPACE "${database}" DATAFILE '/var/oracle_db/orclvm/${database}' SIZE 10M
 AUTOEXTEND ON NEXT 6M MAXSIZE UNLIMITED LOGGING EXTENT MANAGEMENT
 LOCAL SEGMENT SPACE MANAGEMENT AUTO;
CREATE TEMPORARY TABLESPACE "${database}.TEMP" TEMPFILE '/var/oracle_db/orclvm/
${database}.temp' SIZE 5M AUTOEXTEND ON NEXT 5M MAXSIZE UNLIMITED EXTENT
 MANAGEMENT LOCAL UNIFORM SIZE 1M;
CREATE USER "${username}" PROFILE "DEFAULT" IDENTIFIED BY "${password}" DEFAULT
 TABLESPACE "${database}" TEMPORARY TABLESPACE "${database}.TEMP" ACCOUNT
 UNLOCK;
GRANT CREATE SEQUENCE TO "${username}";
GRANT CREATE TABLE TO "${username}";
```

```
GRANT CREATE TRIGGER TO "${username}";
GRANT UNLIMITED TABLESPACE TO "${username}";
GRANT "CONNECT" TO "${username}";
GRANT "RESOURCE" TO "${username}";
```

Security Service

1 Overview

The purpose is to make a simple, unified way for the authentication and the storing/propagation of user sessions through all the eXo components and J2EE containers. JAAS is supposed to be the primary login mechanism but the Security Service framework should not prevent other (custom or standard) mechanisms from being used. You can learn more about JAAS in the *Java Tutorial* [http://java.sun.com/j2se/1.5.0/docs/guide/security/jaas/tutorials/GeneralAcnAndAzn.html]

1 Framework

The central point of this framework is the **ConversationState** object which stores all information about the state of the current user (very similar to the Session concept). The same ConversationState also stores acquired attributes of an **Identity** which is a set of principals to identify a user.

The ConversationState has definite lifetime. This object should be created when the user's identity becomes known by eXo (login procedure) and destroyed when the user leaves an eXo based application (logout procedure). Using JAAS it should happen in LoginModule's login() and logout() methods respectively.

1.1 ConversationState and ConversationRegistry

The ConversationState can be stored

- In a static **local thread variable**, or
- As a **key-value pair** in the **ConversationRegistry** component.

One or the other, or both methods can be used to set/retrieve the state at runtime. The most important thing is that they should be complementary, i.e. make sure that the conversation state is set before you try to use it.

Local Thread Variable: Storing the ConversationState in a static local thread variable makes it possible to represent it as a **context** (current user's state).

```
ConversationState.setCurrent(conversationState);
....
ConversationState.getCurrent();
```

Key-Value way

If you store the ConversationState inside the **ConversationRegistry** component as a set of key-value pairs, the session key is an arbitrary String (user name, ticket id, httpSessionId etc).

```
conversationRegistry.register("key", conversationState);
...
conversationRegistry.getState("key");
```

ConversationRegistry The ConversationRegistry is a mandatory component deployed into eXo Container as following:

```
<component>
   <type>org.exoplatform.services.security.ConversationRegistry</type>
</component>
```

1.1 Authenticator

An Authenticator is responsible for Identity creating, it contains two methods:

- validateUser() accepts an array of credentials and returns the userId (which can be something different from the username).
- createIdentity() accepts the userId and returns a newly created Identity object.

```
public interface Authenticator {
  /**
   * Authenticate user and return userId which can be different to username.
   * @param credentials - list of users credentials (such as name/password, X509 certificate etc)
   * @return userId
   * @throws LoginException
   * @throws Exception
   */
  String validateUser(Credential[] credentials) throws LoginException, Exception;

  /**
   * @param credentials - userId.
   * @return Identity
   * @throws Exception
   */
  Identity createIdentity(String userId) throws Exception;
}
```

It is up to the application developer (and deployer) whether to use the Authenticator component(s) and how many implementations of this components should be deployed in eXo container. The

developer is free to create an Identity object using a different way, but the Authenticator component is the highly recommended way from architectural considerations.

Typical functionality of the validateUser(Credential\[] credentials) method is the comparison of incoming credentials (username/password, digest etc) with those credentials that are stored in an implementation specific database. Then, validateUser(Credential\[] credentials) returns back the userId or throws a LoginException in a case of wrong credentials.

Default Authenticator implementation is org.exoplatform.services.organization.auth.OrganizationAuthenticatorImpl which compares incoming username/password credentials with the ones stored in OrganizationService. Configuration example:

```
<component>
  <key>org.exoplatform.services.security.Authenticator</key>
  <type>org.exoplatform.services.organization.auth.OrganizationAuthenticatorImpl</type>
</component>
```

Usage

JAAS login module

The framework described is not coupled with any authentication mechanism but the most logical and implemented by default is the JAAS Login module. The typical sequence looks as follows (see org.exoplatform.services.security.jaas.DefaultLoginModule):

- LoginModule.login() creates a list of credentials using standard JAAS Callbacks features, obtains an Authenticator instance, and creates an Identity object calling Authenticator.authenticate(..) method

```
Authenticator authenticator = (Authenticator) container()
        .getComponentInstanceOfType(Authenticator.class);
// RolesExtractor can be null
RolesExtractor rolesExtractor = (RolesExtractor) container().
getComponentInstanceOfType(RolesExtractor.class);


Credential[] credentials = new Credential[] {new UsernameCredential(username), new
 PasswordCredential(password) };
String userId = authenticator.validateUser(credentials);
identity = authenticator.createIdentity(userId);
```

- LoginModule.commit() obtains the IdentityRegistry object, and register the identity using userId as a key.

When initializing the login module, you can set the option parameter "singleLogin". With this option you can disallow the same Identity to login for a second time.

By default singleLogin is disabled, so the same identity can be registered more than one time. Parameter can be passed in this form singleLogin=yes or singleLogin=true.

```
IdentityRegistry identityRegistry = (IdentityRegistry)
 getContainer().getComponentInstanceOfType(IdentityRegistry.class);

if (singleLogin && identityRegistry.getIdentity(identity.getUserId()) != null)
  throw new LoginException("User " + identity.getUserId() + " already logined.");

identity.setSubject(subject);
identityRegistry.register(identity);
```

In the case of using several LoginModules, JAAS allows to place the login() and commit() methods in different REQUIRED modules.

After that, the web application must use SetCurrentIdentityFilter. This filter obtains the ConversationRegistry object and tries to get the ConversationState by sessionId (HttpSession). If there is no ConversationState, then SetCurrentIdentityFilter will create a new one, register it and set it as current one using ConversationState.setCurrent(state).

- LoginModule.logout() can be called by JAASConversationStateListener, it extends ConversationStateListener.

This listener must be configured in web.xml. The method sessionDestroyed(HttpSessionEvent) is called by the ServletContainer. This method removes the ConversationState from the ConversationRegistry ConversationRegistry.unregister(sesionId) and calls the method LoginModule.logout().

```
ConversationRegistry conversationRegistry = (ConversationRegistry)
 getContainer().getComponentInstanceOfType(ConversationRegistry.class);

ConversationState conversationState = conversationRegistry.unregister(sesionId);

if (conversationState != null) {
  log.info("Remove conversation state " + sesionId);
  if (conversationState.getAttribute(ConversationState.SUBJECT) != null) {
    Subject subject = (Subject) conversationState.getAttribute(ConversationState.SUBJECT);
    LoginContext ctx = new LoginContext("exo-domain",  subject);
```

```
    ctx.logout();
} else {
  log.warn("Subject was not found in ConversationState attributes.");
}
```

1.1 Predefinded JAAS login modules

There are several JAAS Login modules included in eXo Platform sources:

org.exoplatform.services.security.jaas.DefaultLoginModule which provides both authentication (using eXo Authenticator based mechanism) and authorization, filling Conversation Registry as it described in previous section. There are also several per-Application Server extensions of this login module which can be found in org.exoplatform.services.security.jaas package, which can be used in appropriate AS. In particular, we have dedicated Login modules for Tomcat, JBoss, Jonas and WebSphere.

Besides that, for the case when third party authentication mechanism required, we have **org.exoplatform.services.security.jaas.IdentitySetLoginModule**, which catches a login identity from third party "authenticating" login module and preforms eXo specific authorization job. In this case third party login module has to put login (user) name to the shared state map under **"javax.security.auth.login.name"** key and third party LM has to be configured before IdentitySetLoginModule like:

```
exo {
   com.third.party.LoginModuleImpl required;
   org.exoplatform.services.security.jaas.IdentitySetLoginModule required;
};
```

1.1 J2EE container authentication

As you know, when a user in JAAS is authenticated, a Subject is created as a result. This Subject represents the authenticated user. It is important to know and follow the rules regarding Subject filling which are specific for each J2EE server, where eXo Platform is deployed.

To make it workable for the particular J2EE server, it is necessary to add specific Principals/ Credentials to the Subject to be propagated into the specific J2EE container implementation. We extended the DefaultLoginModule by overloading its commit() method with a dedicated logic, presently available for Tomcat, JBOSS and JONAS application servers.

Furthermore, you can use the optional RolesExtractor which is responsible for mapping primary Subject's principals (userId and a set of groups) to J2EE Roles:

```
public interface RolesExtractor {
```

```
  Set <String> extractRoles(String userId, Set<MembershipEntry> memberships);
}
```

This component may be used by Authenticator to create the Identity with a particular set of **Roles**.

Spring Security Integration

Introduction

How to Integrate the spring security framework in the eXo portal?

This tutorial will guide you through a few steps and show you how easy it is to integrate spring security (or the Spring framework in general) in eXo portal. We will create a login portlet example as a support all along the document reading. The login portlet example has been developed and deployed using the eXo WCM product running on the application server JBoss 4.2.3. But it can easily be adapted to another eXo product (such as ECM) and to other servers such as tomcat. Moreover, the example, claiming to be a real world example, is implemented using JSF 1.2, the JBoss portlet bridge and Spring and can serve as a example project from where you can start your own portlet development targeting the eXo platform.

Installation

This tutorial assumes that you have a working eXo WCM installation running under JBoss 4.2.x.

Download the spring framework: *http://s3.amazonaws.com/dist.springframework.org/release/SPR/spring-framework-2.5.6-with-dependencies.zip*

Download spring-security: *http://sourceforge.net/project/showfiles.php?group_id=73357&package_id=270072&release_id=630203*

Unzip the 02portal.war file in the jboss server/default/deploy/exoplatform.sar directory and copy the following jars in WEB-INF/lib:

- spring.jar
- spring-security-core.jar
- aspectjrt-1.5.4.jar
- exo-spring.jar (contains the filters and event handlers described in this tutorial - see the attachment section of this page)

Configuration

To enable spring security in exo we need to go through a few configuration steps:

JAAS disabling

First, we need to disable the JAAS security which is the default authentication mechanism in exo. Edit 02portal.war web.xml file and comment out the JAAS configuration related lines:

```
...
```

```
  <session-config>
    <session-timeout>15</session-timeout>
  </session-config>

  <!--
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>user authentication</web-resource-name>
      <url-pattern>/private/*</url-pattern>
      <http-method>POST</http-method>
      <http-method>GET</http-method>
    </web-resource-collection>
    <auth-constraint>
      <role-name>users</role-name>
    </auth-constraint>
    <user-data-constraint>
      <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
  </security-constraint>

  <login-config>
    <auth-method>FORM</auth-method>
    <realm-name>exo-domain</realm-name>
    <form-login-config>
      <form-login-page>/login/jsp/login.jsp</form-login-page>
      <form-error-page>/login/jsp/login.jsp</form-error-page>
    </form-login-config>
  </login-config>
   -->

  <security-role>
    <description>a simple user role</description>
    <role-name>users</role-name>
  </security-role>
...
```

Enabling spring security

To enable spring and set the spring security filter, add the following lines:

```
...
  <context-param>
    <param-name>contextConfigLocation</param-name>
```

```
    <param-value>/WEB-INF/security-context.xml</param-value>
  </context-param>

  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>

  <filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
  </filter>
...
```

Activate the spring security filter at the right position, i.e. just after the filter responsible of exo container initialization.

```
...
  <filter-mapping>
    <filter-name>PortalContainerInitializedFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <filter-mapping>
    <filter-name>SetCurrentIdentityFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
...
```

security-context.xml

We need to configure the spring security filter chain for our purposes. Create a file named security-context.xml in 02portal.war WEB-INF directory containing the following lines:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security"
  xmlns:beans="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
                xsi:schemaLocation="http://www.springframework.org/schema/beans http://
www.springframework.org/schema/beans/spring-beans-2.0.xsd
                http://www.springframework.org/schema/security http://www.springframework.org/
schema/security/spring-security-2.0.1.xsd">

  <http auto-config="true">
    <intercept-url pattern="/private/**" access="ROLE_USER" />
    <form-login login-page='/public/classic/Login' default-target-url='/private/classic/home' />
  </http>

  <authentication-provider>
    <user-service>
      <user name="rod" password="koala" authorities="ROLE_SUPERVISOR, ROLE_USER,
 ROLE_TELLER" />
      <user name="root" password="exo" authorities="ROLE_USER" />
    </user-service>
  </authentication-provider>

</beans:beans>
```

The file contains two elements. The http node which is responsible of configuring the filter chain. The auto-config mode set to true allows us to do just a minimal configuration, everything else being smartly initialized by default. We just set an intercept URL pointing to '/private/**' with the ROLE_USER authority which corresponds to secured resources in exo. In case of successful auhentication, the user will be redirected to the specified default target URL.

The second element defines a simple authentication provider based on the spring security InMemoryDaoImpl implementation of the UserDetailsService. Note that we define the exo root user in the configuration which will allow us to log in with admin privileges in the exo portal.

Login portlet example

Now that we have successfully installed and configured spring security in exo, we need a login portlet example to capture user credentials and serve as an entry point in the authentication process. The login portlet itself is based on JSF 1.2, Jboss portlet bridge and the spring framework, but you can obviously use whatever web framework you want to achieve the same.

Building the portlet

So we need a login form to capture user credentials inputs. The portlet login form consists of the following lines of xml:

```
<f:view xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://java.sun.com/jsf/html"
```

```
  xmlns:ice="http://www.icesoft.com/icefaces/component"
  xmlns:liferay-faces="http://liferay.com/tld/faces"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:c="http://java.sun.com/jstl/core"
  xmlns:fn="http://java.sun.com/jsp/jstl/functions"
  xmlns:t="http://myfaces.apache.org/tomahawk">

  <style type="text/css" media="screen">
    @import "/loginportlet/css/starter.css";
    @import "/loginportlet/css/uni-form.css";
  </style>

  <script src="/loginportlet/js/jquery.js" type="text/javascript"></script>
  <script src="/loginportlet/js/uni-form.jquery.js" type="text/javascript"></script>

  <h:form styleClass="uniForm" >
    <fieldset class="inlineLabels">
      <legend>Sign in</legend>

      <div class="ctrlHolder">
        <h:outputLabel for="login" style="width: 70px"><em>*</em>Login:</h:outputLabel>
         <h:inputText id="login" value="#{loginBean.login}" required="true" styleClass="textInput"
/>
        <h:message for="login" styleClass="portlet-msg-error" />
      </div>
      <div class="ctrlHolder">
                  <h:outputLabel  for="password"  style="width:  70px"><em>*</em>Password:</
h:outputLabel>
                    <h:inputSecret  id="password"  value="#{loginBean.passwd}"  required="true"
 styleClass="textInput" />
        <h:message for="password" styleClass="portlet-msg-error" />
      </div>
    </fieldset>

    <div class="buttonHolder" style="margin-top: 20px; margin-right: 20px">
      <h:commandButton styleClass="primaryAction" value="Submit" action="#{loginBean.login}"
 />
    </div>
  </h:form>
</f:view>
```

The interesting part resides in the backing bean which implements the login action triggered when the user clicks the login form submit button.

```
package org.exoplatform.loginportlet;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Controller;

@Controller
@Scope("request")
public class LoginBean {

   String login;

   String passwd;

   public String login() throws Exception {
       String redirect = "/portal/j_spring_security_check?j_username=" + login + "&j_password="
 + passwd;
      PortalUtils.sendRedirect(redirect);
      return null;
   }

   ...
}
```

The login action simply sends a HTTP redirect to the spring security login URL passing the user login and password as parameters. This URL informs the filter to try to authenticate the supplied user credentials. This is the Spring security authentication process entry point.

Setting up the login portal page

Now that we have a login portlet available we need to set it up into a portal page.

- Log in as root in exo portal.
- Go to application registry and import the loginportlet
- Add a new hidden page named 'Login' under the portal classic's navigation (read more on page creation *here>WCM.Tutorial*). Make sure that the visible flag is unchecked to hide the page. Also declare the page as public so that everyone can access it without being authenticated for obvious reasons.
- Finally, drag & drop the login portlet in the page with the desired layout.

Customization of portal login and logout urls

In the portal header, there is a login or logout action displayed depending whether you are already logged in or not. We need to customize those actions so that when the user clicks on it she or he

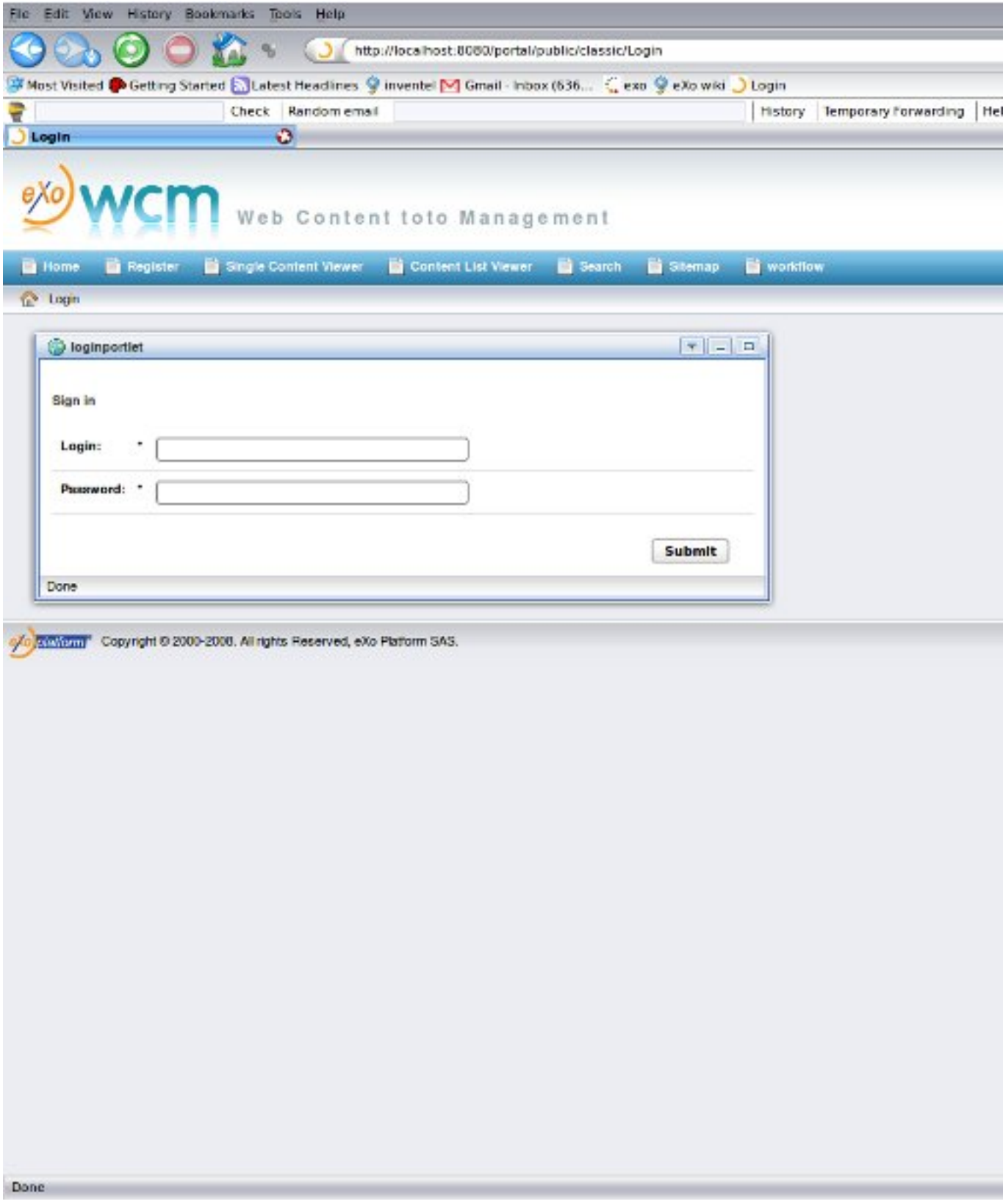
will be redirected either to our login page or to the spring security logout url. Edit the article, go to the default.js tab and apply the following changes to the code:

```
function validateUser() {

   var user = eXo.env.portal.userName;
   var rootObj = document.getElementById("classic-access");
         var loginContentObj = eXo.core.DOMUtil.findFirstDescendantByClass(rootObj, "div",
"UIWCMLoginPortlet");
          var welcomeObj = eXo.core.DOMUtil.findFirstDescendantByClass(rootObj, "span",
"Welcome");
   var userObj = eXo.core.DOMUtil.findFirstDescendantByClass(rootObj, "span", "LoggedUser");
           var languageObj = eXo.core.DOMUtil.findFirstDescendantByClass(rootObj, "a",
"LanguageIcon");
   var logXXXObj = eXo.core.DOMUtil.findPreviousElementByTagName(languageObj, "a");

   if (user != "null") {
      welcomeObj.innerHTML = "Welcome: ";
      userObj.innerHTML = user;
      logXXXObj.innerHTML = "Logout";
      if (eXo.core.DOMUtil.hasClass(logXXXObj, "LoginIcon")) {
         eXo.core.DOMUtil.removeClass(logXXXObj, "LoginIcon");
         eXo.core.DOMUtil.addClass(logXXXObj, "LogoutIcon");
      }
      logXXXObj.onclick = function() { document.location.href = '/portal/j_spring_security_logout' }
   } else {
      if (eXo.core.DOMUtil.hasClass(logXXXObj, "LogoutIcon")) {
         eXo.core.DOMUtil.removeClass(logXXXObj, "LogoutIcon");
         eXo.core.DOMUtil.addClass(logXXXObj, "LoginIcon");
      }
      logXXXObj.innerHTML = "Login";
      logXXXObj.onclick = function() { document.location.href = '/portal/public/classic/Login' };
   }

       languageObj.onclick = function () { if(document.getElementById('UIMaskWorkspace'))
ajaxGet(eXo.env.server.createPortalURL('UIPortal', 'ChangeLanguage', true)); }
}

eXo.core.Browser.addOnLoadCallback("validateUser", validateUser);
```

As you can see, the two onclick event handler function bodies have been changed to a simple redirect to the login page or the logout URL.

A look at the login page

Once you are done with all this, just click on the login action and you should be redirect to the login page looking something like that:


File Edit View History Bookmarks Tools Help
http://localhost:8080/portal/public/classic/Login
Most Visited Getting Started Latest Headlines inventel Gmail - Inbox (636... exo eXo wiki Login
Check Random email History Temporary Forwarding
Login
eXo wcm Web Content toto Management
Home Register Single Content Viewer Content List Viewer Search Sitemap workflow
Login
loginportlet
Sign in
Login:
Password:
Submit
Done
Copyright © 2000-2008. All rights Reserved, eXo Platform SAS.
Done

Integration strategies

Until now we haven't discussed about any integration strategies concerning a potential existing security realm outside of the eXo platform. To address this problem we have the choice between at least two different strategies:

1.1.1 **Direct integration** We can directly integrate eXo with the external realm. Everything related to organisation and user management in exo is cleanly separated in its own abstraction accessible through the OrganisationService. The authentication process itself is encapsulated in the Authenticator abstraction which sits on top of the organization service. eXo provides several implementations of both. So whether your realm is based on LDAP or JDBC and because the default implementations are generic enough, you will be able to use them and fits them to your needs with a matter of a little configuration. You can even develop a custom implementation to meet your more specific needs.

Replication

Or we can go through a replication process between the external realm and the eXo platform realm. This is the strategy that we are going to use to build our login portlet example. Furthermore, the replication will occur dynamically on any user authentication attempt.

Integration with eXo portal

Being successfully authenticated against an external realm is not sufficient by itself. We also need to propagate the newly created security context to the portal own security mechanism. In eXo portal terminology, it means we have to create an Identity object for the user and register it into the Identity Registry.

Spring framework provides a simple notification model where a bean can listen to application events published by other beans. Fortunately, spring security uses this mechanism and publishes an InteractiveAuthenticationSuccessEvent in case of successful authentication. That will allow us to hook up custom code to that event.

Furthermore, we need to replicate the user details from the external realm to the eXo portal one according to the integration strategy defined above.

We create a SpringSecurityEventHandler bean that implements the ApplicationListener interface and listens to the InteractiveAuthenticationSuccessEvent event.

```
package org.exoplatform.spring.security.web;

...

public class SpringSecurityEventHandler implements ApplicationListener {

    private String portalContainerName = "portal";
```

```
  public void onApplicationEvent(ApplicationEvent event) {
    if (event instanceof InteractiveAuthenticationSuccessEvent) {
      try {
                                        InteractiveAuthenticationSuccessEvent successEvent =
(InteractiveAuthenticationSuccessEvent) event;
        ExoContainer container = getContainer();

        String login = successEvent.getAuthentication().getName();
        String passwd = successEvent.getAuthentication().getCredentials().toString();

                                          IdentityRegistry identityRegistry = (IdentityRegistry)
container.getComponentInstanceOfType(IdentityRegistry.class);
                                             Authenticator authenticator = (Authenticator)
container.getComponentInstanceOfType(Authenticator.class);
                                    OrganizationService orgService = (OrganizationService)
container.getComponentInstanceOfType(OrganizationService.class);

        User user = orgService.getUserHandler().findUserByName(login);
        if (user == null) {
          user = orgService.getUserHandler().createUserInstance(login);
          user.setFirstName(login);
          user.setLastName(login);
          orgService.getUserHandler().createUser(user, false);
          orgService.getUserHandler().saveUser(user, false);
          //TODO: put some more integration code here
        }

        Identity identity = authenticator.createIdentity(login);

        Subject subject = new Subject();
        subject.getPrivateCredentials().add(passwd);
        subject.getPublicCredentials().add(new UsernameCredential(login));

        identity.setSubject(subject);
        identityRegistry.register(identity);

      } catch (Exception e) {
        e.getMessage();
      }
    }
  }

  protected ExoContainer getContainer() {
```

```
        // TODO set correct current container
        ExoContainer container = ExoContainerContext.getCurrentContainer();
        if (container instanceof RootContainer) {
            container = RootContainer.getInstance().getPortalContainer(portalContainerName);
        }
        return container;
    }

...

}
```

Basically the bean retrieves user login and password from the InteractiveAuthenticationSuccessEvent object and tries to get the user from the organization service. In case he cannot find it in the repository, he simply creates it on the fly. In this example the user is created with just a few details, but you can put some custom integration code with the external realm here, and create the user with all the details (email, birth date, roles, etc.) it seems appropriate to you. After that, the bean creates an Identity object with the help of the authenticator service, populates it with a subject containing the user credentials and registers it. That's all we have to do to make the portal aware of the user logging in.

Registering our bean is done the usual way in security-context.xml file:

```
...
<beans:bean                                                    id="myEventHandler"
 class="org.exoplatform.spring.security.web.SpringSecurityEventHandler" />
...
```

Security context propagation to portlets

Part of the problem is the question of security context propagation between on one side the portal webapp and at the other side the portlets webapps. This means that the security context has to be available in the portlet side allowing the application logic to deal the with current user principal and granted authorities. By default, Spring security uses a thread local variable to partially achieve this. But a problem may arise due to the fact that the portal invokes the portlet through a webapp cross context call. This means that it can lead to a class cast exceptions (two different classloaders involved), or that the security context is simply not propagated at all. To accommodate this, we will need to set up two request filters, one at the portal webapp side and the other at the portlet webapp side and use the http request to propagate the context in between.

Portal side filter

We will use the spring security extensible filter chain to plug in our filter.

```
package org.exoplatform.spring.security.web;

...

public class PortalSideSecurityContextFilter extends SpringSecurityFilter {

   @Override
      protected void doFilterHttp(HttpServletRequest request, HttpServletResponse response,
 FilterChain chain) throws IOException, ServletException {
      //fill request with security context
      SecurityContext context = SecurityContextHolder.getContext();


 context);

       //fill request with security last exception
                                                                        Object e =
 request.getSession().getAttribute(AbstractProcessingFilter.SPRING_SECURITY_LAST_EXCEPTION_KEY);


 e);

      chain.doFilter(request, response);
   }

   public int getOrder() {
      // TODO Auto-generated method stub
      return 0;
   }
}
```

The PortalSideSecurityContextFilter simply fills the request with the security context and security last exception using the HttpSessionContextIntegrationFilter.SPRING_SECURITY_CONTEXT_KEY and AbstractProcessingFilter.SPRING_SECURITY_LAST_EXCEPTION_KEY attribute names. The portlet can have a look to the AbstractProcessingFilter.SPRING_SECURITY_LAST_EXCEPTION_KEY attribute to check if a security exception has occured.

The following lines in the security-context file register our custom filter in the chain at the last position.

```
...
```

```
                                                    <beans:bean                    id="myCustomFilter"
 class="org.exoplatform.spring.security.web.PortalSideSecurityContextFilter">
     <custom-filter after="LAST" />
   </beans:bean>
...
```

Portlet side filter

In the portlet webapp we create a regular filter named PortletSideSecurityContextFilter.

```
package org.exoplatform.spring.security.web;

...

public class PortletSideSecurityContextFilter implements Filter {

    public void destroy() {

    }

   public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain)
 throws IOException, ServletException {
                                                                    Object          object          =
 request.getAttribute(HttpSessionContextIntegrationFilter.SPRING_SECURITY_CONTEXT_KEY);
        SecurityContext context = (SecurityContext) serializeDeserialize(object);
        if (context != null) {
            SecurityContextHolder.setContext(context);
        } else {
            SecurityContextHolder.clearContext();
        }

        filterChain.doFilter(request, response);
    }

    public void init(FilterConfig arg0) throws ServletException {

    }

    private Object serializeDeserialize(Object obj) {
        Object result = null;
        try {
            ByteArrayOutputStream bout = new ByteArrayOutputStream();
            ObjectOutputStream out = new ObjectOutputStream(bout);
```

```
            out.writeObject(obj);

            ByteArrayInputStream bin =    new ByteArrayInputStream(bout.toByteArray());
            ObjectInputStream in = new ObjectInputStream(bin);

            result = in.readObject();
        } catch (Exception e) {
            //TODO: handle exception
        }
        return result;
    }

}
```

The PortletSideSecurityContextFilter retrieves the security context from the request and proceeds to a serialization de-serialization of it to avoid a potential class cast exception that may occur when propagating an object across webapps. Then the context is simply set or cleared whether the context is null or not.

To register your filter simply add the following lines to your portlet webapp web.xml file.

```
...
  <filter>
    <filter-name>portletSideSecurityContextFilter</filter-name>
          <filter-class>org.exoplatform.spring.security.web.PortletSideSecurityContextFilter</filter-
class>
  </filter>

  <filter-mapping>
    <filter-name>portletSideSecurityContextFilter</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
    <dispatcher>FORWARD</dispatcher>
  </filter-mapping>
...
```

Conclusion

We are done! Now we know how to integrate the spring security framework in the eXo portal. Thanks to the the great integration capabilities of both eXo portal and Spring framework. You can have a look to the attachment section on this page and get the source code of this tutorial.

Organization Service

Overview

OrganizationService is the service that allows to access the Organization model. This model is composed of :

- Users
- Groups
- Memberships

It is the basis of eXo personalization and authorizations in eXo and is used to all over the platform. The model is abstract and does not rely on any specific storage. Multiple implementations exist in eXo :

- Hibernate : for storage into a RDBMS
- Jndi : for storage into a directory such as an LDAP or MS Active Directory
- Jcr : for storage inside a Java Content Repository

Organizational Model

User

- Username used as the identified
- Profile (identity and preferences)

Group

Gather a set of users

- Applicative or business
- Tree structure
- No inheritance
- Expressed as /group/subgroup/subsubgroup

Membership

- Qualifies the group belonging

- "Member of group as XXX"
- Expressed as : manager:/organization/hr, *:/partners

Related articles and how-tos

- *JCR Organization Service* [http://wiki.exoplatform.org/xwiki/bin/view/JCR/Organization+Service]
- *Update ConversationState when user's Membership changed* [http://wiki.exoplatform.org/xwiki/bin/view/Core/Update+ConversationState+when+user's+Membership+changed]
- *Organization Service Initializer* [Organization Service Initializer]
- *How to Access User Profile in your code* [http://wiki.exoplatform.org/xwiki/bin/view/Portal/Accessing+User+Profile]
- *How to create your own Organization Listener* [CoreOrganizationListener]
- *How to manipulate Users and Memberships Programmatically* [How to manipulate Users and Memberships Programmatically]

Organization Service Initializer

Use the Organization Service Initializer to create users, groups and membership types by default.

```
<external-component-plugins>
            <target-component>org.exoplatform.services.organization.OrganizationService</target-
component>
   <component-plugin>
     <name>init.service.listener</name>
     <set-method>addListenerPlugin</set-method>
     <type>org.exoplatform.services.organization.OrganizationDatabaseInitializer</type>
     <description>this listener populate organization data for the first launch</description>
     <init-params>
       <value-param>
         <name>checkDatabaseAlgorithm</name>
         <description>check database</description>
         <value>entry</value>
       </value-param>
       <value-param>
         <name>printInformation</name>
         <description>Print information init database</description>
         <value>false</value>
       </value-param>
       <object-param>
         <name>configuration</name>
         <description>description</description>
         <object type="org.exoplatform.services.organization.OrganizationConfig">
           <field name="membershipType">
             <collection type="java.util.ArrayList">
               <value>
                                                                                    <object
 type="org.exoplatform.services.organization.OrganizationConfig$MembershipType">
                   <field name="type">
                     <string>manager</string>
                   </field>
                   <field name="description">
                     <string>manager membership type</string>
                   </field>
                 </object>
               </value>
             </collection>
           </field>
```

```
        <field name="group">
          <collection type="java.util.ArrayList">
            <value>
              <object type="org.exoplatform.services.organization.OrganizationConfig$Group">
                <field name="name">
                  <string>platform</string>
                </field>
                <field name="parentId">
                  <string></string>
                </field>
                <field name="description">
                  <string>the /platform group</string>
                </field>
                <field name="label">
                  <string>Platform</string>
                </field>
              </object>
            </value>
            <value>
              <object type="org.exoplatform.services.organization.OrganizationConfig$Group">
                <field name="name">
                  <string>administrators</string>
                </field>
                <field name="parentId">
                  <string>/platform</string>
                </field>
                <field name="description">
                  <string>the /platform/administrators group</string>
                </field>
                <field name="label">
                  <string>Administrators</string>
                </field>
              </object>
            </value>
          </collection>
        </field>

        <field name="user">
          <collection type="java.util.ArrayList">
            <value>
              <object type="org.exoplatform.services.organization.OrganizationConfig$User">
                <field name="userName">
                  <string>root</string>
                </field>
```

```
                <field name="password">
                  <string>exo</string>
                </field>
                <field name="firstName">
                  <string>Root</string>
                </field>
                <field name="lastName">
                  <string>Root</string>
                </field>
                <field name="email">
                  <string>root@localhost</string>
                </field>
                <field name="groups">
                  <string>
                    manager:/platform/administrators
                  </string>
                </field>
              </object>
            </value>
          </collection>
        </field>
      </object>
    </object-param>
  </init-params>
 </component-plugin>
</external-component-plugins>
```

Params for membership type:

- type: The membership type's name.
- description: The membership type's description.

Params for group:

- name: The group's name
- parentId: The id of the parent group. If the parent id is null, it means that the group is at the first level. The parentId should have the form: /ancestor/parent
- description: The group's description
- label: The group's label

Params for user:

- userName: The user's name
- password: The user's password
- firstName: The user's first name
- lastName: The user's last name
- email: The user's email
- groups: The user's membership types and groups in which he consist.

Organization Listener

Overview

The *Organization Service* provides a mechanism to receive notifications when :

- A User is created, deleted or modified.
- A Group is created, deleted or modified.
- A Membership is created or removed.

This mechanism is very useful to cascade some actions when the organization model is modified. For example, it is currently used to :

- Initialize the personal portal pages.
- Initialize the personal calendars, address books and mail accounts in CS.
- Create drives and personal areas in ECM.

Writing your own listeners

To implement your own listener, you just need to write extend some existing listener classes. These classes define hooks that are invoked before or after operations are performed on organization model.

UserEventListener

To listen to user changes, you need to extend <>org.exoplatform.services.organization.UserEventListener</> :

```
public class MyUserListener extends UserEventListener {

  public void preSave(User user, boolean isNew) throws Exception {
    System.out.println("Before " + (isNew?"creating":"updating") + " user " + user.getUserName());
  }

  public void postSave(User user, boolean isNew) throws Exception {
    System.out.println("After user " + user.getUserName() + (isNew?" created":" updated"));
  }

  public void preDelete(User user) throws Exception {
    System.out.println("Before deleting user " + user.getUserName());
```

```
  }

  public void preDelete(User user) throws Exception {
    System.out.println("After deleting user " + user.getUserName());
  }

}
```

GroupEventListener

To listen to group changes, you need to extend <>org.exoplatform.services.organization.GroupEventListener</> :

```
public class MyGroupListener extends GroupEventListener {

  public void preSave(Group group, boolean isNew) throws Exception {
    System.out.println("Before " + (isNew?"creating":"updating") + " group " + group.getName());
  }

  public void postSave(Group group, boolean isNew) throws Exception {
    System.out.println("After group " + group.getName() + (isNew?" created":" updated"));
  }

  public void preDelete(Group group) throws Exception {
    System.out.println("Before deleting group " + group.getName());
  }

  public void preDelete(Group group) throws Exception {
    System.out.println("After deleting group " + group.getName());
  }
}
```

MembershipEventListener

To listen to membership changes, you need to extend <>org.exoplatform.services.organization.MembershipEventListener</> :

```
public class MyMembershipListener extends MembershipEventListener {

  public void preSave(Membership membership, boolean isNew) throws Exception {
    System.out.println("Before " + (isNew?"creating":"updating") + " membership.");
  }
```

```
  public void postSave(Membership membership, boolean isNew) throws Exception {
    System.out.println("After membership " + (isNew?" created":" updated"));
  }

  public void preDelete(Membership membership) throws Exception {
    System.out.println("Before deleting membership");
  }

  public void preDelete(Membership membership) throws Exception {
    System.out.println("After deleting membership");
  }
}
```

Registering your listeners

Registering the listeners is then achieved by using the ExoContainer plugin mechanism. Learn more about it on the *Service Configuration for Beginners* article.

To effectively register organization service's listeners you simply need to use the <>addListenerPlugin</> seer injector.

So, the easiest way to register your listeners is to pack them into a .jar and create a configuration file into it under **mylisteners.jar!/conf/portal/configuration.xml**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>
 <external-component-plugins>
            <target-component>org.exoplatform.services.organization.OrganizationService</target-component>
   <component-plugin>
    <name>myuserplugin</name>
    <set-method>addListenerPlugin</set-method>
    <type>org.example.MyUserListener</type>
    <description></description>
   </component-plugin>
   <component-plugin>
    <name>mygroupplugin</name>
    <set-method>addListenerPlugin</set-method>
    <type>org.example.MyGroupListener</type>
    <description></description>
   </component-plugin>
   <component-plugin>
    <name>mymembershipplugin</name>
```

```
    <set-method>addListenerPlugin</set-method>
    <type>org.example.MyMembershipListener</type>
    <description></description>
   </component-plugin>
  </external-component-plugins>
<configuration>
```

Now, simply deploy the jar under $TOMCAT_HOME/lib and your listeners are ready!

Update ConversationState when user's Membership changed

When a user logged in portal in ConversationRegistry added ConversationSate for this user. ConversationState keeps user's Identity that is actual for logged in time. In this case even user's Membership updated in OrganizationService ConversationState still keeps old (not actual Identity). User must logged out and loggin in again to update Identity. To fix this issue, need add special listener in configuration of OrganizationServicer. This listener is extended MembershipEventListener.

Example of configuration.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd      http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
 <external-component-plugins>
          <target-component>org.exoplatform.services.organization.OrganizationService</target-
component>
.....
.....
   <component-plugin>
     <name>MembershipUpdateListener</name>
     <set-method>addListenerPlugin</set-method>
     <type>org.exoplatform.services.organization.impl.MembershipUpdateListener</type>
   </component-plugin>

  <external-component-plugins>
</configuration>
```

DB Schema creator service (JDBC implementation)

DB Schema Creator is responsible for creating database schema, using a DDL script inside service configuration or in an external file, calling:

```
org.exoplatform.services.database.jdbc.DBSchemaCreator.createTables(String dsName, String script)
```

via

```
org.exoplatform.services.database.jdbc.CreateDBSchemaPlugin component plugin
```

A configuration example:

```
<component>
  <key>org.exoplatform.services.database.jdbc.DBSchemaCreator</key>
  <type>org.exoplatform.services.database.jdbc.DBSchemaCreator</type>
  <component-plugins>
    <component-plugin>
      <name>jcr.dbschema</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.database.jdbc.CreateDBSchemaPlugin</type>
      <init-params>
        <value-param>
          <name>data-source</name>
          <value>jdbcjcr</value>
        </value-param>
        <value-param>
          <name>script-file</name>
          <value>conf/storage/jcr-mjdbc.sql</value>
        </value-param>
      </init-params>
    </component-plugin>
 .....
```

An example of a DDL script:

```
CREATE TABLE JCR_MITEM(
      ID VARCHAR(255) NOT NULL PRIMARY KEY,
      VERSION INTEGER NOT NULL,
      PATH VARCHAR(1024) NOT NULL
      );
CREATE INDEX JCR_IDX_MITEM_PATH ON JCR_MITEM(PATH);
```

Database Configuration for Hibernate

As usual, it is quite simple to use our configuration XML syntax to configure and parametrize different Databases for eXo tables but also for your own use.

Generic configuration

The default DB configuration uses HSQLDB, a Java Database quite useful for demonstrations.

```
<component>
  <key>org.exoplatform.services.database.HibernateService</key>
  <jmx-name>exo-service:type=HibernateService</jmx-name>
  <type>org.exoplatform.services.database.impl.HibernateServiceImpl</type>
  <init-params>
    <properties-param>
      <name>hibernate.properties</name>
      <description>Default Hibernate Service</description>
      <property name="hibernate.show_sql" value="false"/>
      <property name="hibernate.cglib.use_reflection_optimizer" value="true"/>
      <property name="hibernate.connection.url" value="jdbc:hsqldb:file:../temp/data/portal"/>
      <property name="hibernate.connection.driver_class" value="org.hsqldb.jdbcDriver"/>
      <property name="hibernate.connection.autocommit" value="true"/>
      <property name="hibernate.connection.username" value="sa"/>
      <property name="hibernate.connection.password" value=""/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect"/>
      <property name="hibernate.c3p0.min_size" value="5"/>
      <property name="hibernate.c3p0.max_size" value="20"/>
      <property name="hibernate.c3p0.timeout" value="1800"/>
      <property name="hibernate.c3p0.max_statements" value="50"/>
    </properties-param>
  </init-params>
</component>
```

In the init parameter section, we define the default hibernate properties including the DB URL, the driver and the credentials in use.

For any portals that configuration can be overridden, depending on the needs of your environment.

Several databases have been tested and can be used in production....which is not the case of HSQLDB, HSQLDB can only be used for development environments and for demonstrations.

Example DB configuration

For MySQL

```
<component>
  <key>org.exoplatform.services.database.HibernateService</key>
  <jmx-name>database:type=HibernateService</jmx-name>
  <type>org.exoplatform.services.database.impl.HibernateServiceImpl</type>
  <init-params>
    <properties-param>
      <name>hibernate.properties</name>
      <description>Default Hibernate Service</description>
      <property name="hibernate.show_sql" value="false"/>
      <property name="hibernate.cglib.use_reflection_optimizer" value="true"/>
                                <property     name="hibernate.connection.url"     value="jdbc:mysql://
localhost:3306/
exodb?relaxAutoCommit=true&amp;amp;autoReconnect=true&amp;amp;useUnicode=true&amp;amp;characterEnc
>
      <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
      <property name="hibernate.connection.autocommit" value="true"/>
      <property name="hibernate.connection.username" value="exo"/>
      <property name="hibernate.connection.password" value="exo"/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
      <property name="hibernate.c3p0.min_size" value="5"/>
      <property name="hibernate.c3p0.max_size" value="20"/>
      <property name="hibernate.c3p0.timeout" value="1800"/>
      <property name="hibernate.c3p0.max_statements" value="50"/>
    </properties-param>
  </init-params>
</component>
```

Registering custom Hibernate XML files into the service

It is possible to use the eXo hibernate service and register your hibernate hbm.xml files to leverage some add-on features of the service such as the table automatic creation as well as the cache of the hibernate session in a ThreadLocal object during all the request lifecycle. To do so, you just have to add a plugin and indicate the location of your files.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration>
 <external-component-plugins>
  <target-component>org.exoplatform.services.database.HibernateService</target-component>
```

```
    <component-plugin>
      <name>add.hibernate.mapping</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.database.impl.AddHibernateMappingPlugin</type>
      <init-params>
        <values-param>
          <name>hibernate.mapping</name>
          <value>org/exoplatform/services/organization/impl/UserImpl.hbm.xml</value>
          <value>org/exoplatform/services/organization/impl/MembershipImpl.hbm.xml</value>
          <value>org/exoplatform/services/organization/impl/GroupImpl.hbm.xml</value>
         <value>org/exoplatform/services/organization/impl/MembershipTypeImpl.hbm.xml</value>
          <value>org/exoplatform/services/organization/impl/UserProfileData.hbm.xml</value>
        </values-param>
      </init-params>
    </component-plugin>
  </external-component-plugins>
</configuration>
```

LDAP Configuration

Overview

You may decide that you want eXo users to be mapped to an existing directory. eXo provides a flexible implementation of its OrganizationService on top of LDAP. It can be used on any LDAP compliant directory and even Active Directory. This page will guide you how to configure eXo Platform to work with your directory.

Quickstart

If you just want to have a look at how eXo works with ldap. eXo comes with a predefined ldap configuration. You just need to activate it and eXo will create all it needs to work at startup.

You need to have a working ldap server and a user with write permissions.

- Open **exo-tomcat/webapps/portal/WEB-INF/conf/configuration.xml** and replace:

```
<import>war:/conf/organization/hibernate-configuration.xml</import>
```

With

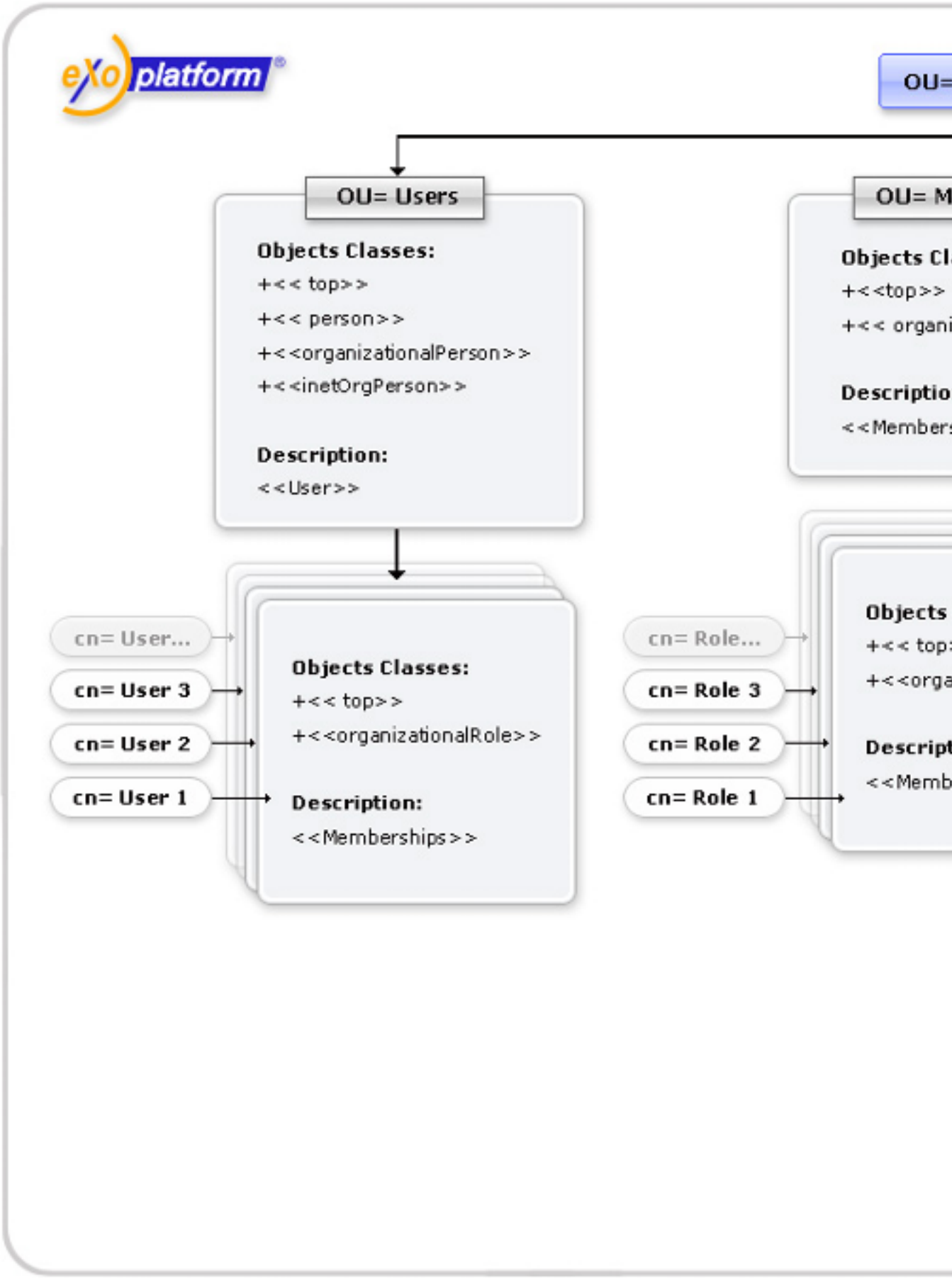
```
<import>war:/conf/organization/ldap-configuration.xml</import>
```

- 0pen **ldap-configuration.xml** and update the **providerURL**, **rootdn** and password settings according to your environment

```
<field name="providerURL"><string>ldap://127.0.0.1:389</string></field>
<field name="rootdn"><string>CN=Manager,DC=MyCompany,DC=com</string></field>
<field name="password"><string>secret</string></field>
```

- Delete **exo-tomcat/temp/*** to have a clean database and then start tomcat.

eXo starts and autocreates its organization model in your directory tree. Finally, the structure of the default LDAP schema looks like:


OU=
OU= Users
Objects Classes:
+<< top>>
+<< person>>
+<<organizationalPerson>>
+<<inetOrgPerson>>
Description:
<<User>>
OU= M
Objects Cl
+<<top>>
+<< organi
Descriptio
<<Members
cn= User...
cn= User 3
cn= User 2
cn= User 1
Objects Classes:
+<< top>>
+<<organizationalRole>>
Description:
<<Memberships>>
cn= Role...
cn= Role 3
cn= Role 2
cn= Role 1
Objects
+<< top
+<<orga
Descript
<<Memb

That's it! Now eXo uses your LDAP directory as its org model storage. Users, groups and memberships are now stored and retrieved from there. We suggest that you complete some guideline functions with eXo user management portlet and see what it changes in your directory tree.

Configuration

If you have an existing LDAP server, the eXo predefined settings will likely not match your directory structure. eXo LDAP organization service implementation was written with flexibility in mind and can certainly be configured to meet your requirements.

The configuration is done in **ldap-configuration.xml** file, and this chapter will explain the numerous parameters it contains.

Connection Settings

Firstly, start by connection settings which will tell eXo how to connect to your directory server. These settings are very close to *JNDI API* [http://java.sun.com/products/jndi] context parameters. This configuration is activated by the init-param ldap.config of service LDAPServiceImpl.

```
<component>
  <key>org.exoplatform.services.ldap.LDAPService</key>
  <type>org.exoplatform.services.ldap.impl.LDAPServiceImpl</type>
  <init-params>
    <object-param>
      <name>ldap.config</name>
      <description>Default ldap config</description>
      <object type="org.exoplatform.services.ldap.impl.LDAPConnectionConfig">
        <field name="providerURL"><string>ldap://127.0.0.1:389,10.0.0.1:389</string></field>
        <field name="rootdn"><string>CN=Manager,DC=exoplatform,DC=org</string></field>
        <field name="password"><string>secret</string></field>
        <!-- field  name="authenticationType"><string>simple</string></field-->
        <field name="version"><string>3</string></field>
        <field  name="referralMode"><string>follow</string></field>
        <!-- field  name="serverName"><string>active.directory</string></field-->
        <field name="minConnection"><int>5</int></field>
        <field name="maxConnection"><int>10</int></field>
        <field name="timeout"><int>50000</int></field>
      </object>
    </object-param>
  </init-params>
</component>
```

- **providerURL**: LDAP server URL (see *PROVIDER_URL* [http://java.sun.com/products/jndi/1.2/javadoc/javax/naming/Context.html#PROVIDER_URL]). For multiple ldap servers, use comma separated list of host:port (Ex. ldap://127.0.0.1:389,10.0.0.1:389).
- **rootdn**: dn of user that will be used by the service to authenticate on the server (see *SECURITY_PRINCIPAL* [http://java.sun.com/products/jndi/1.2/javadoc/javax/naming/Context.html#SECURITY_PRINCIPAL]).
- **password**: password for user rootdn (see *SECURITY_CREDENTIALS* [http://java.sun.com/products/jndi/1.2/javadoc/javax/naming/Context.html#SECURITY_CREDENTIALS]).
- **authenticationType**: type of authentication to be used (see *SECURITY_AUTHENTICATION* [http://java.sun.com/products/jndi/1.2/javadoc/javax/naming/Context.html#SECURITY_AUTHENTICATION]). Use one of none, simple, strong. Default is simple.
- **version**: LDAP protocol version (see *java.naming.ldap.version* [http://java.sun.com/products/jndi/tutorial/ldap/misc/version.html]). Set to 3 if your server supports LDAP V3.
- **referalMode**: one of follow, ignore,throw (see *REFERRAL* [http://java.sun.com/products/jndi/1.2/javadoc/javax/naming/Context.html#REFERRAL]).
- **serverName**: you will need to set this to active.directory in order to work with Active Directory servers. Any other value will be ignore and the service will act as on a standard LDAP.
- **maxConnection**: the maximum number of connections per connection identity that can be maintained concurrently.
- **minConnection**: the number of connections per connection identity to create when initially creating a connection for the identity.
- **timeout**: the number of milliseconds that an idle connection may remain in the pool without being closed and removed from the pool.

Organization Service Configuration

Next, you need to configure the eXo **OrganizationService** to tell him how the directory is structured and how to interact with it. This is managed by a couple of of init-params : **ldap.userDN.key** and **ldap.attribute.mapping** in file **ldap-configuration.xml** (by default located at portal.war/WEB-INF/conf/organization)

```
<component>
  <key>org.exoplatform.services.organization.OrganizationService</key>
  <type>org.exoplatform.services.organization.ldap.OrganizationServiceImpl</type>
  [...]
  <init-params>
    <value-param>
      <name>ldap.userDN.key</name>
```

```
    <description>The key used to compose user DN</description>
    <value>cn</value>
   </value-param>
   <object-param>
    <name>ldap.attribute.mapping</name>
    <description>ldap attribute mapping</description>
    <object type="org.exoplatform.services.organization.ldap.LDAPAttributeMapping">
    [...]
   </object-param>
  </init-params>
  [...]
</component>
```

ldap.attribute.mapping maps your ldap to eXo. At first there are two main parameters to configure in it:

```
<field name="baseURL"><string>dc=exoplatform,dc=org</string></field>
<field name="ldapDescriptionAttr"><string>description</string></field>
```

- **baseURL**: root dn for eXo organizational entities. This entry can't be created by eXo and must preexist in directory.
- **ldapDescriptionAttr** (since core 2.2+) : Name of a common attribute that will be used as description for groups and membership types.

Note

(since core 2.2+) : Name of a common attribute that will be used as description for groups and membership types.

Other parameters are discussed in the following sections.

Users

Main parameters

Here are the main parameters to map eXo users to your directory :

```
<field name="userURL"><string>ou=users,ou=portal,dc=exoplatform,dc=org</string></field>
<field name="userObjectClassFilter"><string>objectClass=person</string></field>
<field name="userLDAPClasses"><string>top,person,organizationalPerson,inetOrgPerson</
string></field>
```

- **userURL** : base dn for users. Users are created in a flat structure under this base with a dn of the form: **ldap.userDN.key=username,userURL**

Example :

```
uid=john,cn=People,o=MyCompany,c=com
```

However, if users exist deeply under userURL, eXo will be able to retrieve them.

Example :

```
uid=tom,ou=France,ou=EMEA,cn=People,o=MyCompany,c=com
```

- **userObjectClassFilter**: Filter used under userURL branch to distinguish eXo user entries from others.

Example : john and tom will be recognized as valid eXo users but EMEA and France entries will be ignored in the following subtree :

```
uid=john,cn=People,o=MyCompany,c=com
 objectClass: person
 …
ou=EMEA,cn=People,o=MyCompany,c=com
 objectClass: organizationalUnit
 …
  ou=France,ou=EMEA,cn=People,o=MyCompany,c=com
   objectClass: organizationalUnit
   …
    uid=tom,ou=EMEA,cn=People,o=MyCompany,c=com
     objectClass: person
     …
```

- **userLDAPClasses** : comma separated list of classes used for user creation.

When creating a new user, an entry will be created with the given objectClass attributes. The classes must at least define cn and any attribute refernced in the user mapping.

Example : Adding the user Marry Simons could produce :

```
uid=marry,cn=users,ou=portal,dc=exoplatform,dc=org
 objectclass: top
```

```
 objectClass: person
 objectClass: organizationalPerson
 objectClass: inetOrgPerson
 …
```

User mapping

The following parameters maps ldap attributes to eXo User java objects attributes.

```
<field name="userUsernameAttr"><string>uid</string></field>
<field name="userPassword"><string>userPassword</string></field>
<field name="userFirstNameAttr"><string>givenName</string></field>
<field name="userLastNameAttr"><string>sn</string></field>
<field name="userDisplayNameAttr"><string>displayName</string></field>
<field name="userMailAttr"><string>mail</string></field>
```

- **userUsernameAttr**: username (login)
- **userPassword**: password (used when portal authentication is done by eXo login module)
- **userFirstNameAttr**: firstname
- **userLastNameAttr**: lastname
- **userDisplayNameAttr**: displayed name
- **userMailAttr**: email address

Example : In the previous example, user Marry Simons could produce :

```
uid=marry,cn=users,ou=portal,dc=exoplatform,dc=org
 objectclass: top
 objectClass: person
 objectClass: organizationalPerson
 objectClass: inetOrgPerson
 …
```

Groups

eXo groups can be mapped to organizational or applicative groups defined in your directory.

```
<field name="groupsURL"><string>ou=groups,ou=portal,dc=exoplatform,dc=org</string></
field>
```

```
<field name="groupLDAPClasses"><string>top,organizationalUnit</string></field>
<field name="groupObjectClassFilter"><string>objectClass=organizationalUnit</string></field>
```

- **groupsURL** : base dn for eXo groups

Groups can be structured hierarchically under groupsURL.

Example: Groups communication, communication/marketing and communication/press would map to :

```
ou=communication,ou=groups,ou=portal,dc=exoplatform,dc=org
…
 ou=marketing,ou=communication,ou=groups,ou=portal,dc=exoplatform,dc=org
 …
 ou=press,ou=communication,ou=groups,ou=portal,dc=exoplatform,dc=org
 …
```

- **groupLDAPClasses**: comma separated list of classes used for group creation.

When creating a new group, an entry will be created with the given objectClass attributes. The classes must define at least the required attributes: **ou**, **description** and **l**.

Note

l attribute corresponds to the City property in OU property editor

Example : Adding the group human-resources could produce:

```
ou=human-resources,ou=groups,ou=portal,dc=exoplatform,dc=org
 objectclass: top
 objectClass: organizationalunit
 ou: human-resources
 description: The human resources department
 l: Human Resources
 …
```

- **groupObjectClassFilter**: filter used under groupsURL branch to distinguish eXo groups from other entries. You can also use a complex filter if you need.

Example : groups WebDesign, WebDesign/Graphists and Sales could be retrieved in :

```
l=Paris,dc=sites,dc=mycompany,dc=com
 …
```

```
 ou=WebDesign,l=Paris,dc=sites,dc=mycompany,dc=com
 …
  ou=Graphists,WebDesign,l=Paris,dc=sites,dc=mycompany,dc=com
  …
l=London,dc=sites,dc=mycompany,dc=com
 …
 ou=Sales,l=London,dc=sites,dc=mycompany,dc=com
 …
```

Membership Types

Membership types are the possible roles that can be assigned to users in groups.

```
<field

string></field>
<field name="membershipTypeLDAPClasses"><string>top,organizationalRole</string></field>
<field name="membershipTypeNameAttr"><string>cn</string></field>
```

- **membershipTypeURL** : base dn for membership types storage.

eXo stores membership types in a flat structure under membershipTypeURL.

Example : Roles manager, user, admin and editor could by defined by the subtree :

```
ou=roles,ou=portal,dc=exoplatform,dc=org
…
 cn=manager,ou=roles,ou=portal,dc=exoplatform,dc=org
 …
 cn=user,ou=roles,ou=portal,dc=exoplatform,dc=org
 …
 cn=admin,ou=roles,ou=portal,dc=exoplatform,dc=org
 …
 cn=editor,ou=roles,ou=portal,dc=exoplatform,dc=org
 …
```

- **membershipTypeLDAPClasses**: comma separated list of classes for membership types creation.

When creating a new membership type, an entry will be created with the given objectClass attributes. The classes must define the required attributes : **description**, **cn**

Example : Adding membership type validator would produce :

```
cn=validator,ou=roles,ou=portal,dc=exoplatform,dc=org
  objectclass: top
  objectClass: organizationalRole
  …
```

- **membershipTypeNameAttr** : Attribute that will be used as the name of the role

Example : If membershipTypeNameAttr is 'cn', then role name is 'manager' for the following membership type entry :

```
cn=manager,ou=roles,ou=portal,dc=exoplatform,dc=org </pre>
```

Memberships

Memberships are used to assign a role within a group. They are entries that are placed under the group entry of their scope group. Users in this role are defined as attributes of the membership entry.

Example: To designate tom as the manager of the group human-resources:

```
ou=human-resources,ou=groups,ou=portal,dc=exoplatform,dc=org
  …
  cn=manager,ou=human-resources,ou=groups,ou=portal,dc=exoplatform,dc=org
    member: uid=tom,ou=users,ou=portal,dc=exoplatform,dc=org
    …
```

The parameters to configure memberships are:

```
<field name="membershipLDAPClasses"><string>top,groupOfNames</string></field>
<field name="membershipTypeMemberValue"><string>member</string></field>
<field name="membershipTypeRoleNameAttr"><string>cn</string></field>
<field      name="membershipTypeObjectClassFilter"><string>objectClass=organizationalRole</
string></field>
```

- **membershipLDAPClasses** : comma separated list of classes used to create memberships.

When creating a new membership, an entry will be created with the given objectClass attributes. The classes must at least define the attribute designated by membershipTypeMemberValue.

Example : Adding membership validator would produce :

```
cn=validator,ou=human-resources,ou=groups,ou=portal,dc=exoplatform,dc=org
  objectclass: top
  objectClass: groupOfNames
  …
```

<pre> cn=validator,ou=human-resources,ou=groups,ou=portal,dc=exoplatform,dc=org objectclass: top objectClass: groupOfNames ... </pre>

- **membershipTypeMemberValue**: Multivalued attribute used in memberships to reference users that have the role in the group.

Values should be a user dn.

Example: james and root have admin role within the group human-resources, would give:

```
cn=admin,ou=human-resources,ou=groups,ou=portal,dc=exoplatform,dc=org
  member: cn=james,ou=users,ou=portal,dc=exoplatform,dc=org
  member: cn=root,ou=users,ou=portal,dc=exoplatform,dc=org
  …
```

- **membershipTypeRoleNameAttr**: Attribute of the membership entry whose value references the membership type.

Example : In the following membership entry:

<pre> cn=manager,ou=human-resources,ou=groups,ou=portal,dc=exoplatform,dc=org </pre>

'cn' attribute is used to designate the 'manager' membership type. Which could also be said : The name of the role is given by 'cn' the attribute.

- **membershipTypeObjectClassFilter** : Filter used to distinguish membership entries under groups.

You can use rather complex filters.

Example: Here is a filter we used for a customer that needed to trigger a dynlist overlay on openldap.

```
(&amp;(objectClass=ExoMembership)(membershipURL=*))
```

Note: Pay attention to the xml escaping of the '&' (and) operator

User Profiles

eXo User profiles also have entries in the ldap but the actual storage is still done with the hibernate service. You will need the following parameters:

```
<field name="profileURL"><string>ou=profiles,ou=portal,dc=exoplatform,dc=org</string></field>
<field name="profileLDAPClasses"><string>top,organizationalPerson</string></field>
```

- **profileURL**: base dn to store user profiles
- **profileLDAPClasses**: Classes used to when creating user profiles

Advanced topics

Automatic directory population

At startup, eXo can populate the organization model based on

eXo organizational model has User,Group,Membership and Profile entities. For each, we define a base dn that should be below baseURL. At startup, if one of userURL, groupsURL, membershipTypeURL or profileURL does not exist fully, eXo will attempt to create the missing subtree by parsing the dn and creating entries on-the-fly. To determine the classes of the created entries, the following rules are applied :

- ou=... : objectClass=top,objectClass=organizationalUnit
- cn=... : objectClass=top,objectClass=organizationalRole
- c=... : objectClass=country
- o=... : objectClass=organization
- dc=.. : objectClass=top,objectClass=dcObject,objectClass=organization

Example:

If baseURL is **o=MyCompany,c=com** and groupsURL is **dc=groups,cn=Extranet,c=France,ou=EMEA,o=MyCompany,c=com** then, the following subtree will be created :

```
ou=EMEA,o=MyCompany,c=com
 objectClass: top
 objectClass: organizationalUnit
 …
  c=France,ou=EMEA,o=MyCompany,c=com
    objectClass: top
    objectClass: country
```

```
…
cn=Extranet,c=France,ou=EMEA,o=MyCompany,c=com
  objectClass: top
  objectClass: organizationalRole
  …
  dc=groups,cn=Extranet,c=France,ou=EMEA,o=MyCompany,c=com
    objectClass: top
    objectClass: dcObject
    objectClass: organization
    …
```

Active Directory sample configuration

Here is an alternative configuration for active directory that you can find in **activedirectory-configuration.xml**

Note

There is a microsoft limitation: password can't be set in AD via unsecured connection you have to use the ldaps protocol

here is how to use LDAPS protocol with Active Directory :

```
1 setup AD to use SSL:

   * add Active Directory Certificate Services role
   * install right certificate for DC machine

2 enable Java VM to use certificate from AD:

   * import root CA used in AD, to keystore, something like

      keytool -importcert -file 2008.cer -keypass changeit -keystore /home/user/java/jdk1.6/jre/lib/security/cacerts

   * set java options

                JAVA_OPTS="${JAVA_OPTS}  -Djavax.net.ssl.trustStorePassword=changeit -Djavax.net.ssl.trustStore=/home/user/java/jdk1.6/jre/lib/security/cacerts"
```

```
[...]
  <component>
  <key>org.exoplatform.services.ldap.LDAPService</key>
```

```
[..]
      <object type="org.exoplatform.services.ldap.impl.LDAPConnectionConfig">
            <!-- for multiple ldap servers, use comma seperated list of host:port (Ex. ldap://
127.0.0.1:389,10.0.0.1:389) -->
     <!-- whether or not to enable ssl, if ssl is used ensure that the javax.net.ssl.keyStore &
 java.net.ssl.keyStorePassword properties are set -->
  <!-- exo portal default installed javax.net.ssl.trustStore with file is java.home/lib/security/cacerts-
->
    <!-- ldap service will check protocol, if protocol is ldaps, ssl is enable (Ex. for enable ssl:
 ldaps://10.0.0.3:636 ;for disable ssl: ldap://10.0.0.3:389 ) -->
   <!-- when enable ssl, ensure server name is *.directory and port (Ex. active.directory) -->
   <field  name="providerURL"><string>ldaps://10.0.0.3:636</string></field>
      <field    name="rootdn"><string>CN=Administrator,CN=Users,  DC=exoplatform,DC=org</
string></field>
   <field  name="password"><string>site</string></field>
   <field  name="version"><string>3</string></field>
     <field  name="referralMode"><string>ignore</string></field>
     <field  name="serverName"><string>active.directory</string></field>
      </object>
[..]
 <component>
  <key>org.exoplatform.services.organization.OrganizationService</key>
  [...]
     <object type="org.exoplatform.services.organization.ldap.LDAPAttributeMapping">
       [...]
       <field  name="userAuthenticationAttr"><string>mail</string></field>
       <field  name="userUsernameAttr"><string>sAMAccountName</string></field>
       <field  name="userPassword"><string>unicodePwd</string></field>
       <field  name="userLastNameAttr"><string>sn</string></field>
       <field  name="userDisplayNameAttr"><string>displayName</string></field>
       <field  name="userMailAttr"><string>mail</string></field>
       [..]
       <field  name="membershipTypeLDAPClasses"><string>top,group</string></field>
        <field  name="membershipTypeObjectClassFilter"><string>objectClass=group</string></
field>
       [..]
       <field  name="membershipLDAPClasses"><string>top,group</string></field>
       <field  name="membershipObjectClassFilter"><string>objectClass=group</string></field>
     </object>
     [...]
</component>
```

OpenLDAP dynlist overlays

If you use OpenLDAP, you may want to use the *overlays* [http://www.openldap.org/faq/data/cache/1169.html]. Here is how you can use the *dynlist overlay* [http://www.openldap.org/faq/data/cache/1209.html] to have memberships dynamically populated.

The main idea is to have your memberships populated dynamically by an ldap query. Thus, you no longer have to maintain manually the roles on users.

To configure the dynlist, add the following to your **slapd.conf** :

```
dynlist-attrset         ExoMembership membershipURL member
```

This snipet means : On entries that have ExoMembership class, use the URL defined in the value of attribute membershipURL as a query and populate results under the multivalues attribute member.

Now let's declare the corresponding schema (replace XXXXX to adapt to your own IANA code):

```
attributeType ( 1.3.6.1.4.1.XXXXX.1.59 NAME 'membershipURL' SUP memberURL )
```

membershipURL inherits from memberURL.

```
objectClass ( 1.3.6.1.4.1.XXXXX.2.12   NAME 'ExoMembership' SUP top MUST ( cn ) MAY (membershipURL $ member $ description ) )
```

ExoMembership must define cn and can have attributes :

- membershipURL: trigger for the dynlist
- member : attribute populated by the dynlist
- description : used by eXo for display

```
# the TestGroup group
dn: ou=testgroup,ou=groups,ou=portal,o=MyCompany,c=com
objectClass: top
objectClass: organizationalUnit
ou: testgroup
l: TestGroup
description: the Test Group
```

On this group, we can bind an eXo membership where the overlay will occur:

```
# the manager membership on group TestGroup
dn: cn=manager, ou=TestGroup,ou=groups,ou=portal,o=MyCompany,c=com
objectClass: top
objectClass: ExoMembership
membershipURL: ldap:///ou=users,ou=portal,o=MyCompany,c=com??sub?(uid=*)
cn: manager
```

This dynlist assigns the role **manager:/testgroup** to any user.

Tika Document Reader Service

Intro

DocumentReaderService provides API to retrieve DocumentReader by mimetype.

DocumentReader lets the user fetch content of document as String or, in case of TikaDocumentReader, as Reader.

Architecture

Basicaly, DocumentReaderService is a container for all registered DocumentReaders. So, you can register DocumentReader (method addDocumentReader(ComponentPlugin reader)) and fetch DocumentReader by mimeType (method getDocumentReader(String mimeType)).

TikaDocumentReaderServiceImpl extends DocumentReaderService with simple goal - read Tika configuration and lazy register each Tika Parser as TikaDocumentReader.

Note

By default, all Tikas Parsers are not registered in readers <mimetype, DocumentReader> map. When user tries to fetch a DocumentReader by unknown mimetype. Than TikaDocumentReaderService checks tika configuration, and register a new mimetype-DocumentReader pair.

Configuration

How TikaDocumentReaderService Impl configuration looks like:

```
<component>
    <key>org.exoplatform.services.document.DocumentReaderService</key>
   <type>org.exoplatform.services.document.impl.tika.TikaDocumentReaderServiceImpl</type>

    <!-- Old-style document readers -->
    <component-plugins>
      <component-plugin>
        <name>pdf.document.reader</name>
        <set-method>addDocumentReader</set-method>
        <type>org.exoplatform.services.document.impl.PDFDocumentReader</type>
        <description>to read the pdf inputstream</description>
      </component-plugin>

      <component-plugin>
        <name>document.readerMSWord</name>
        <set-method>addDocumentReader</set-method>
```

```
    <type>org.exoplatform.services.document.impl.MSWordDocumentReader</type>
    <description>to read the ms word inputstream</description>
  </component-plugin>

  <component-plugin>
    <name>document.readerMSXWord</name>
    <set-method>addDocumentReader</set-method>
    <type>org.exoplatform.services.document.impl.MSXWordDocumentReader</type>
    <description>to read the ms word inputstream</description>
  </component-plugin>

  <component-plugin>
    <name>document.readerMSExcel</name>
    <set-method>addDocumentReader</set-method>
    <type>org.exoplatform.services.document.impl.MSExcelDocumentReader</type>
    <description>to read the ms excel inputstream</description>
  </component-plugin>

  <component-plugin>
    <name>document.readerMSXExcel</name>
    <set-method>addDocumentReader</set-method>
    <type>org.exoplatform.services.document.impl.MSXExcelDocumentReader</type>
    <description>to read the ms excel inputstream</description>
  </component-plugin>

  <component-plugin>
    <name>document.readerMSOutlook</name>
    <set-method>addDocumentReader</set-method>
    <type>org.exoplatform.services.document.impl.MSOutlookDocumentReader</type>
    <description>to read the ms outlook inputstream</description>
  </component-plugin>

  <component-plugin>
    <name>PPTdocument.reader</name>
    <set-method>addDocumentReader</set-method>
    <type>org.exoplatform.services.document.impl.PPTDocumentReader</type>
    <description>to read the ms ppt inputstream</description>
  </component-plugin>

  <component-plugin>
    <name>MSXPPTdocument.reader</name>
    <set-method>addDocumentReader</set-method>
    <type>org.exoplatform.services.document.impl.MSXPPTDocumentReader</type>
    <description>to read the ms pptx inputstream</description>
```

```
    </component-plugin>

    <component-plugin>
      <name>document.readerHTML</name>
      <set-method>addDocumentReader</set-method>
      <type>org.exoplatform.services.document.impl.HTMLDocumentReader</type>
      <description>to read the html inputstream</description>
    </component-plugin>

    <component-plugin>
      <name>document.readerXML</name>
      <set-method>addDocumentReader</set-method>
      <type>org.exoplatform.services.document.impl.XMLDocumentReader</type>
      <description>to read the xml inputstream</description>
    </component-plugin>

    <component-plugin>
      <name>TPdocument.reader</name>
      <set-method>addDocumentReader</set-method>
      <type>org.exoplatform.services.document.impl.TextPlainDocumentReader</type>
      <description>to read the plain text inputstream</description>
      <init-params>
        <!--
       values-param> <name>defaultEncoding</name> <description>description</description>
<value>UTF-8</value>
          </values-param
        -->
      </init-params>
    </component-plugin>

    <component-plugin>
      <name>document.readerOO</name>
      <set-method>addDocumentReader</set-method>
      <type>org.exoplatform.services.document.impl.OpenOfficeDocumentReader</type>
      <description>to read the OO inputstream</description>
    </component-plugin>

  </component-plugins>

  <init-params>
    <value-param>
      <name>tika-configuration</name>
      <value>jar:/conf/portal/tika-config.xml</value>
    </value-param>
```

```
    </init-params>

  </component>
</configuration>
```

tika-config.xml example:

```
<properties>

 <mimeTypeRepository magic="false"/>
 <parsers>

  <parser name="parse-dcxml" class="org.apache.tika.parser.xml.DcXMLParser">
   <mime>application/xml</mime>
   <mime>image/svg+xml</mime>
   <mime>text/xml</mime>
   <mime>application/x-google-gadget</mime>
  </parser>

  <parser name="parse-office" class="org.apache.tika.parser.microsoft.OfficeParser">
   <mime>application/excel</mime>
   <mime>application/xls</mime>
   <mime>application/msworddoc</mime>
   <mime>application/msworddot</mime>
   <mime>application/powerpoint</mime>
   <mime>application/ppt</mime>

   <mime>application/x-tika-msoffice</mime>
   <mime>application/msword</mime>
   <mime>application/vnd.ms-excel</mime>
   <mime>application/vnd.ms-excel.sheet.binary.macroenabled.12</mime>
   <mime>application/vnd.ms-powerpoint</mime>
   <mime>application/vnd.visio</mime>
   <mime>application/vnd.ms-outlook</mime>
  </parser>

  <parser name="parse-ooxml" class="org.apache.tika.parser.microsoft.ooxml.OOXMLParser">
   <mime>application/x-tika-ooxml</mime>
   <mime>application/vnd.openxmlformats-package.core-properties+xml</mime>
   <mime>application/vnd.openxmlformats-officedocument.spreadsheetml.sheet</mime>
   <mime>application/vnd.openxmlformats-officedocument.spreadsheetml.template</mime>
   <mime>application/vnd.ms-excel.sheet.macroenabled.12</mime>
   <mime>application/vnd.ms-excel.template.macroenabled.12</mime>
```

```
    <mime>application/vnd.ms-excel.addin.macroenabled.12</mime>
   <mime>application/vnd.openxmlformats-officedocument.presentationml.presentation</mime>
    <mime>application/vnd.openxmlformats-officedocument.presentationml.template</mime>
    <mime>application/vnd.openxmlformats-officedocument.presentationml.slideshow</mime>
    <mime>application/vnd.ms-powerpoint.presentation.macroenabled.12</mime>
    <mime>application/vnd.ms-powerpoint.slideshow.macroenabled.12</mime>
    <mime>application/vnd.ms-powerpoint.addin.macroenabled.12</mime>
          <mime>application/vnd.openxmlformats-officedocument.wordprocessingml.document</
mime>
    <mime>application/vnd.openxmlformats-officedocument.wordprocessingml.template</mime>
    <mime>application/vnd.ms-word.document.macroenabled.12</mime>
    <mime>application/vnd.ms-word.template.macroenabled.12</mime>
   </parser>

   <parser name="parse-html" class="org.apache.tika.parser.html.HtmlParser">
    <mime>text/html</mime>
   </parser>

   <parser mame="parse-rtf" class="org.apache.tika.parser.rtf.RTFParser">
    <mime>application/rtf</mime>
   </parser>

   <parser name="parse-pdf" class="org.apache.tika.parser.pdf.PDFParser">
    <mime>application/pdf</mime>
   </parser>

   <parser name="parse-txt" class="org.apache.tika.parser.txt.TXTParser">
    <mime>text/plain</mime>
    <mime>script/groovy</mime>
    <mime>application/x-groovy</mime>
    <mime>application/x-javascript</mime>
    <mime>application/javascript</mime>
    <mime>text/javascript</mime>
   </parser>

                                          <parser          name="parse-openoffice"
class="org.apache.tika.parser.opendocument.OpenOfficeParser">

    <mime>application/vnd.oasis.opendocument.database</mime>

    <mime>application/vnd.sun.xml.writer</mime>
    <mime>application/vnd.oasis.opendocument.text</mime>
    <mime>application/vnd.oasis.opendocument.graphics</mime>
    <mime>application/vnd.oasis.opendocument.presentation</mime>
```

```
    <mime>application/vnd.oasis.opendocument.spreadsheet</mime>
    <mime>application/vnd.oasis.opendocument.chart</mime>
    <mime>application/vnd.oasis.opendocument.image</mime>
    <mime>application/vnd.oasis.opendocument.formula</mime>
    <mime>application/vnd.oasis.opendocument.text-master</mime>
    <mime>application/vnd.oasis.opendocument.text-web</mime>
    <mime>application/vnd.oasis.opendocument.text-template</mime>
    <mime>application/vnd.oasis.opendocument.graphics-template</mime>
    <mime>application/vnd.oasis.opendocument.presentation-template</mime>
    <mime>application/vnd.oasis.opendocument.spreadsheet-template</mime>
    <mime>application/vnd.oasis.opendocument.chart-template</mime>
    <mime>application/vnd.oasis.opendocument.image-template</mime>
    <mime>application/vnd.oasis.opendocument.formula-template</mime>
    <mime>application/x-vnd.oasis.opendocument.text</mime>
    <mime>application/x-vnd.oasis.opendocument.graphics</mime>
    <mime>application/x-vnd.oasis.opendocument.presentation</mime>
    <mime>application/x-vnd.oasis.opendocument.spreadsheet</mime>
    <mime>application/x-vnd.oasis.opendocument.chart</mime>
    <mime>application/x-vnd.oasis.opendocument.image</mime>
    <mime>application/x-vnd.oasis.opendocument.formula</mime>
    <mime>application/x-vnd.oasis.opendocument.text-master</mime>
    <mime>application/x-vnd.oasis.opendocument.text-web</mime>
    <mime>application/x-vnd.oasis.opendocument.text-template</mime>
    <mime>application/x-vnd.oasis.opendocument.graphics-template</mime>
    <mime>application/x-vnd.oasis.opendocument.presentation-template</mime>
    <mime>application/x-vnd.oasis.opendocument.spreadsheet-template</mime>
    <mime>application/x-vnd.oasis.opendocument.chart-template</mime>
    <mime>application/x-vnd.oasis.opendocument.image-template</mime>
    <mime>application/x-vnd.oasis.opendocument.formula-template</mime>
  </parser>

  <parser name="parse-image" class="org.apache.tika.parser.image.ImageParser">
    <mime>image/bmp</mime>
    <mime>image/gif</mime>
    <mime>image/jpeg</mime>
    <mime>image/png</mime>
    <mime>image/tiff</mime>
    <mime>image/vnd.wap.wbmp</mime>
    <mime>image/x-icon</mime>
    <mime>image/x-psd</mime>
    <mime>image/x-xcf</mime>
  </parser>

  <parser name="parse-class" class="org.apache.tika.parser.asm.ClassParser">
```

```
    <mime>application/x-tika-java-class</mime>
  </parser>

  <parser name="parse-mp3" class="org.apache.tika.parser.mp3.Mp3Parser">
    <mime>audio/mpeg</mime>
  </parser>

  <parser name="parse-midi" class="org.apache.tika.parser.audio.MidiParser">
    <mime>application/x-midi</mime>
    <mime>audio/midi</mime>
  </parser>

  <parser name="parse-audio" class="org.apache.tika.parser.audio.AudioParser">
    <mime>audio/basic</mime>
    <mime>audio/x-wav</mime>
    <mime>audio/x-aiff</mime>
  </parser>

 </parsers>

</properties>
```

Old-style DocumentReaders and Tika Parsers

As you see configuration above, there is both old-style DocumentReaders and new Tika parsers registered.

But MSWordDocumentReader and org.apache.tika.parser.microsoft.OfficeParser both refer to same "application/msword" mimetype, exclaims attentive reader. And he is right. But only one DocumentReader will be fetched.

Old-style DocumentReader registered in configuration become registered into DocumentReaderService. So, mimetypes that is supported by those DocumentReaders will have a registered pair, and user will always fetch this DocumentReaders with getDocumentReader(..) method. Tika configuration will be checked for Parsers only if there is no already registered DocumentReader.

How to make and register own DocumentReader

You can make you own DocumentReader in two ways.

Old-Style Document Reader:

- extend BaseDocumentReader

```
public class MyDocumentReader extends BaseDocumentReader
{
  public String[] getMimeTypes()
  {
    return new String[]{"mymimetype"};
  }
  ...
}
```

- register it as component-plugin

```
<component-plugin>
  <name>my.DocumentReader</name>
  <set-method>addDocumentReader</set-method>
  <type>com.mycompany.document.MyDocumentReader</type>
  <description>to read my own file format</description>
</component-plugin>
```

Tika Parser:

- implement Parser

```
public class MyParser implements Parser
{
  ...
}
```

- register it in tika-config.xml

```
 <parser name="parse-mydocument" class="com.mycompany.document.MyParser">
     <mime>mymimetype</mime>
 </parser>
```

TikaDocumentReader features and notes

TikaDocumentReader features and notes:

- TikaDocumentReader may return document contant as Reader object. Old-Style DocumentReader does not;

- TikaDocumentReader do not detects document mimetipe. You will get exact parser as configured in tika-config;
- All readers methods closes InputStream at final.

Digest Authentication

Overview

Digest access authentication is one of the agreed methods a web server can use to negotiate credentials with a web user's browser. It uses encryption to send the password over the network which is safer than the Basic access authentication that sends plaintext.

Technically digest authentication is an application of MD5 cryptographic hashing with usage of nonce values to discourage cryptanalysis. It uses the HTTP protocol.

Server configuration

To configure you server to use DIGEST authentication we need to edit serverside JAAS module implementation configuration file.

Tomcat Server configuration

You need to fulfill a couple of steps. Firstly change login configuration:

Edit config file located here: exo-tomcat/webapps/rest.war/WEB-INF/web.xml

Replace

```
<login-config>
  <auth-method>BASIC</auth-method>

  <realm-name>eXo REST services</realm-name>

</login-config>
```

for

```
<login-config>
  <auth-method>DIGEST</auth-method>

  <realm-name>eXo REST services</realm-name>

</login-config>
```

More information about tomcat configuration can be found at *Apache Tomcat Configuration Reference* [http://tomcat.apache.org/tomcat-6.0-doc/config/realm.html].

Secondly you also need to specify new login module for JAAS:

Edit config file located here: exo-tomcat/conf/jaas.conf

Replace

```
exo-domain {
  org.exoplatform.services.security.j2ee.TomcatLoginModule required;
};
```

for

```
exo-domain {
  org.exoplatform.services.security.j2ee.DigestAuthenticationTomcatLoginModule required;
};
```

Jetty server configuration

You need to fulfill a couple of steps. Firstly change login configuration:

Edit config file located here: exo-jetty/webapps/rest.war/WEB-INF/web.xml

Replace

```
<login-config>
  <auth-method>BASIC</auth-method>

  <realm-name>eXo REST services</realm-name>

</login-config>
```

for

```
<login-config>
  <auth-method>DIGEST</auth-method>

  <realm-name>eXo REST services</realm-name>

</login-config>
```

Secondly you also need to specify new login module for JAAS:

Edit config file located here: exo-jetty/jaas.conf

Replace

```
exo-domain {
  org.exoplatform.services.security.j2ee.JettyLoginModule required;
};
```

for

```
exo-domain {
  org.exoplatform.services.security.j2ee.DigestAuthenticationJettyLoginModule required;
};
```

JBoss server configuration

Edit config file located here: exo-jboss/server/default/deploy/exo.jcr.ear.ear/rest.war/WEB-INF/web.xml

Replace

```
<login-config>
  <auth-method>BASIC</auth-method>

  <realm-name>eXo REST services</realm-name>

</login-config>
```

for

```
<login-config>
  <auth-method>DIGEST</auth-method>

  <realm-name>eXo REST services</realm-name>

</login-confi
```

You also need to edit login configuration file located here: exo-jboss/server/default/conf/login-config.xml

```
<application-policy name="exo-domain">
   <authentication>
                                                                        <login-module
 code="org.exoplatform.services.security.j2ee.DigestAuthenticationJbossLoginModule"
                 flag="required">
          <module-option name="usersProperties">props/jmx-console-users.properties</module-
option>
           <module-option name="rolesProperties">props/jmx-console-roles.properties</module-
option>
         <module-option name="hashAlgorithm">MD5</module-option>
         <module-option name="hashEncoding">rfc2617</module-option>
         <module-option name="hashUserPassword">false</module-option>
         <module-option name="hashStorePassword">true</module-option>
         <module-option name="passwordIsA1Hash">true</module-option>
         <module-option name="storeDigestCallback">
            org.jboss.security.auth.spi.RFC2617Digest
         </module-option>
      </login-module>
   </authentication>
</application-policy>
```

You probably should define users.properties and role.properties according to your own needs.

More information about jboss server Digest authentication configuration can be found at *JBoss quide chapter* [http://docs.jboss.org/jbossas/guides/webguide/r2/en/html/ch05.html].

OrganizationService implementation requirements

To make your own `org.exoplatform.services.organization.OrganizationService` implementation able to use DIGEST authentication you need to make your UserHandler implementation also implement `org.exoplatform.services.organization.DigestAuthenticator` interface which provide more flexible authenticate method. As it is called from `org.exoplatform.services.organization.auth.OrganizationAuthenticatorImpl` it receive a `org.exoplatform.services.security.Credential` instances, you can get more information from `org.exoplatform.services.security.PasswordCredential.getPasswordContext()`. It can be used to calculate md5 digest of original password to compare it with recieved from clientside.

Part IV. eXoWS

eXo Web Services

eXo Web Services introduction

The Web Services module allows eXo technology to integrate with external products and services.

It's implementaion of API for RESTful Web Services with extensions, Servlet and cross-domain AJAX web-frameworks and JavaBean-JSON transformer.

Introduction to the Representational State Transfer (REST)

Introduction

Representational State Transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web. The term was introduced in the doctoral dissertation in 2000 by Roy Fielding, one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification, and has come into widespread use in the networking community.

REST strictly refers to a collection of network architecture principles that outline how resources are defined and addressed. The term is often used in a looser sense to describe any simple interface that transmits domain-specific data over HTTP without an additional messaging layer such as SOAP or session tracking via HTTP cookies.

The key abstraction of information in REST is a **resource**. Any information that can be named can be a resource: a document or image, a temporal service (e.g. "today's weather in Los Angeles"), a collection of other resources, a non-virtual object (e.g. a person), and so on. In other words, any concept that might be the target of an author's hypertext reference must fit within the definition of a resource. A resource is a conceptual mapping to a set of entities, not the entity that corresponds to the mapping at any particular point in time.

REST uses a **resource identifier** to identify the particular resource involved in an interaction between components. REST connectors provide a generic interface for accessing and manipulating the value set of a resource, regardless of how the membership function is defined or the type of software that is handling the request. URL or URN are the examples of a resource identifier.

REST components perform actions with a resource by using a **representation** to capture the current or intended state of that resource and transferring that representation between components. A representation is a sequence of bytes, plus **representation metadata** to describe those bytes. Other commonly used but less precise names for a representation include: **document, file, and HTTP message entity, instance, or variant**. A representation consists of data, metadata describing the data, and, on occasion, metadata to describe the metadata (usually for the purpose of verifying message integrity). Metadata are in the form of name-value pairs, where the name corresponds to a standard that defines the value's structure and semantics. The data format of a representation is known as a media type.

Table 82.1. REST Data Elements

Data Element	Modern Web Examples
resource	the intended conceptual target of a hypertext reference

Data Element	Modern Web Examples
resource identifier	URL, URN
representation	HTML document, JPEG image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

REST uses various **connector** types to encapsulate the activities of accessing resources and transferring resource representations. The connectors present an abstract interface for component communication, enhancing simplicity by providing a complete separation of concepts and hiding the underlying implementation of resources and communication mechanisms.

Table 82.2. REST Connectors

Connector	Modern Web Examples
client	libwww, libwww-perl
server	libwww, Apache API, NSAPI
cache	browser cache, Akamai cache network
resolver	bind (DNS lookup library)
tunnel	SOCKS, SSL after HTTP CONNECT

The primary connector types are client and server. The essential difference between the two is that a client initiates communication by making a request, whereas a server listens for connections and responds to requests in order to supply access to its services. A component may include both client and server connectors.

An important part of RESTful architecture is a well-defined interface to communicate, in particular it is a set of HTTP methods such as POST, GET, PUT and DELETE. These methods are often compared with the CREATE, READ, UPDATE, DELETE (CRUD) operations associated with database technologies. An analogy can also be made:

- PUT is analogous to CREATE or PASTE OVER,
- GET to READ or COPY,
- POST to UPDATE or PASTE AFTER, and
- DELETE to DELETE or CUT.

Note: RESTful architecture is not limited to those methods, one of good examples of extension is the WebDAV protocol.

The **CRUD** (Create, Read, Update and Delete) verbs are designed to operate with atomic data within the context of a database transaction. REST is designed around the atomic transfer of a

more complex state and can be viewed as a mechanism for transferring structured information from one application to another.

HTTP separates the notions of a web server and a web browser. This allows the implementation of each to vary from the other based on the client/server principle. When used RESTfully, HTTP is **stateless**. Each message contains all the information necessary to understand the request.

As a result, neither the client nor the server needs to remember any communication-state between messages. Any state retained by the server must be modeled as a resource..

OverwriteDefaultProviders

Motivation

There is set of providers embedded in eXo JAX-RS implementation.

Implementations of MessageBodyReader and MessageBodyWriters are taking care about serialization/deserialization of message body (HTTP request/response's body).

The next set of media and Java types processed automatically, thanks to embedded Readers (Writers).

Table 83.1. Embedded Reader and Writers of message body

Media Type	Java Type
/	byte[]
/	javax.activation.DataSource
/	java.io.File
/	java.io.InputStream
/	java.io.Reader
/	java.lang.String
/	javax.ws.rs.core.StreamingOutput (Writer ONLY)
application/json	1. Object with simple constructor + get/set methods; 2. Java Collection (java.uitl.List<T>, java.uitl.Set<T>, java.util.Map<String, T>, etc) where T as described in 1.
application/x-www-form-urlencoded	javax.ws.rs.core.MultivaluedMap<String, String>
multipart/*	java.util.Iterator<org.apache.commons.fileupload.FileItem>
application/xml, application/ xhtml+xml, text/xml	javax.xml.bind.JAXBElement
application/xml, application/ xhtml+xml, text/xml	Object with JAXB annotations
application/xml, application/ xhtml+xml, text/xml	javax.xml.transform.stream.StreamSource
application/xml, application/ xhtml+xml, text/xml	javax.xml.transform.sax.SAXSource
application/xml, application/ xhtml+xml, text/xml	javax.xml.transform.dom.DOMSource

In some case it may be required to use alternative provider for the same media and java type but such changes must not impact to any other services.

Usage

To be able overwrite default JAX-RS provider(s) developer need:

1. Deploy own RESTful service(s) by using subclass of javax.ws.rs.core.Application (hereinafter Application).
2. Service(s) NOT NEED to implement marker interface ResourceContainer and MUST NOT be configured as component(s) of eXo Container. Instead of it Application must be configured as component of eXo Container.
3. If RESTful services or providers require some dependencies from eXo Container then Application should inject it by own constructor and then delegate to services or providers. As alternative method getClasses() may be used for deliver services/providers classes instead of instances. In this case services/providers will work in per-request mode and RESTful framework will take care about resolving dependencies.

Example

In example below see how to use Jackson JSON provider instead of embedded in eXo RESTful framework.

Create subclass of javax.ws.rs.core.Application with code as bellow and add it to the eXo Container configuration.

```
package org.exoplatform.test.jackson;

import org.codehaus.jackson.jaxrs.JacksonJaxbJsonProvider;

import java.util.HashSet;
import java.util.Set;

import javax.ws.rs.core.Application;

public class Application1 extends Application
{
  @Override
  public Set<Class<?>> getClasses()
  {
    Set<Class<?>> cls = new HashSet<Class<?>>(1);
    cls.add(Resource1.class);
    return cls;
```

```
  }

  @Override
  public Set<Object> getSingletons()
  {
    Set<Object> objs = new HashSet<Object>(1);
    objs.add(new JacksonJaxbJsonProvider());
    return objs;
  }
}
```

In this example we assumes Resource1 is Java class which carries JAX-RS annotations and it uses JSON. In this case RESTful framework will use JacksonJaxbJsonProvider for serializing/ deserializing of all messages to/from Resource1. But it will not impact to other services in any kind.

RestServicesList Service

Overview.

RestServicesList service is intendet to provide information about rest services deployed to the application server.

- Path - path to service
- Regex - service's URL regular expression
- FQN - full qualified name of service's class

The list can be provided in two formats: HTML and JSON.

Usage

Note

Class does not implement org.exoplatform.services.rest.resource.ResourceContainer and must never be binded to RESTful framework by using eXoContainer. This service must works as per-request resource.

HTML format

To get the list of services in HTML format use listHTML() method:

```
@GET
@Produces({MediaType.TEXT_HTML})
public byte[] listHTML()
{
  ...
}
```

To do this, perform a simple GET request to the RestServicesList link.

f.e. curl -u root:exo http://localhost:8080/rest/ will return such HTML code:

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>eXo JAXRS Implementation</title>
  </head>
```

```
  <body>
    <h3 style="text-align:center;">Root resources</h3>
    <table   width="90%"   style="table-layout:fixed;">
      <tr>
        <th>Path</th>
        <th>Regex</th>
        <th>FQN</th>
      </tr>
      <tr>
        <td>script/groovy</td>
        <td>/script/groovy(/.*)?</td>
        <td>org.exoplatform.services.jcr.ext.script.groovy.GroovyScript2RestLoader</td>
      </tr>
      <tr>
        <td>/lnkproducer/</td>
        <td>/lnkproducer(/.*)?</td>
        <td>org.exoplatform.services.jcr.webdav.lnkproducer.LnkProducer</td>
      </tr>
      <tr>
        <td>/registry/</td>
        <td>/registry(/.*)?</td>
        <td>org.exoplatform.services.jcr.ext.registry.RESTRegistryService</td>
      </tr>
      <tr>
        <td>/jcr</td>
        <td>/jcr(/.*)?</td>
        <td>org.exoplatform.services.jcr.webdav.WebDavServiceImpl</td>
      </tr>
      <tr>
        <td>/</td>
        <td>(/.*)?</td>
        <td>org.exoplatform.services.rest.ext.service.RestServicesList</td>
      </tr>
    </table>
  </body>
</html>
```

If you perform the same request with your browser, you'll see the table with the list of deployed services like this:

Table 84.1. Root resources

Path	Regex	FQN
script/groovy	/script/groovy(/.*)?	org.exoplatform.services.jcr.ext.script.groovy.Groo

Path	Regex	FQN
/lnkproducer/	/lnkproducer(/.*)?	org.exoplatform.services.jcr.webdav.lnkproducer.L
/registry/	/registry(/.*)?	org.exoplatform.services.jcr.ext.registry.RESTRegi
/jcr	/jcr(/.*)?	org.exoplatform.services.jcr.webdav.WebDavServi
/	(/.*)?	org.exoplatform.services.rest.ext.service.RestServ

JSON format

To get the list of services in HTML format use listJSON() method:

```
@GET
@Produces({MediaType.APPLICATION_JSON})
public RootResourcesList listJSON()
{
  ...
}
```

To do this, add "Accept:application/json" header to your GET request

f.e. curl -u root:exo http://localhost:8080/rest/ -H "Accept:application/json" will return such JSON:

```
{"rootResources":[
  {
    "fqn":"org.exoplatform.services.jcr.ext.script.groovy.GroovyScript2RestLoader",
    "regex":"/script/groovy(/.*)?",
    "path":"script/groovy"
  },
  {
    "fqn":"org.exoplatform.services.jcr.webdav.lnkproducer.LnkProducer",
    "regex":"/lnkproducer(/.*)?",
    "path":"/lnkproducer/"
  },
  {
    "fqn":"org.exoplatform.services.jcr.ext.registry.RESTRegistryService",
    "regex":"/registry(/.*)?",
    "path":"/registry/"
  },
  {
    "fqn":"org.exoplatform.services.jcr.webdav.WebDavServiceImpl",
    "regex":"/jcr(/.*)?",
    "path":"/jcr"
  },
```

```
  {
    "fqn":"org.exoplatform.services.rest.ext.service.RestServicesList",
    "regex":"(/.*)?",
    "path":"/"
  }
]}
```

Groovy Scripts as REST Services

Overview

This article describes how to use Groovy scripts as REST services. We are going to consider these operations:

- Load script and save it in JCR.
- Instantiate script
- Deploy newly created Class as RESTful service.
- Script Lifecycle Management.
- And finally we will discover simple example which can get JCR node UUID

In this article, we consider RESTful service compatible with JSR-311 specification. Currently last feature available in version 1.11-SNAPSHOT of JCR, 2.0-SNAPSHOT of WS and version 2.1.4-SNAPSHOT of core.

Loading script and save it in JCR

There are two ways to save a script in JCR. The first way is to save it at server startup time by using configuration.xml and the second way is to upload the script via HTTP.

Load script at startup time

This way can be used for load prepared scripts, to use this way. we must configure org.exoplatform.services.jcr.ext.script.groovy.GroovyScript2RestLoaderPlugin. This is simple configuration example.

```
<external-component-plugins>
   <target-component>org.exoplatform.services.jcr.ext.script.groovy.GroovyScript2RestLoader</
target-component>
  <component-plugin>
    <name>test</name>
    <set-method>addPlugin</set-method>
    <type>org.exoplatform.services.jcr.ext.script.groovy.GroovyScript2RestLoaderPlugin</type>
    <init-params>
      <value-param>
        <name>repository</name>
        <value>repository</value>
```

```
      </value-param>
      <value-param>
        <name>workspace</name>
        <value>production</value>
      </value-param>
      <value-param>
        <name>node</name>
        <value>/script/groovy</value>
      </value-param>
      <properties-param>
        <name>JcrGroovyTest.groovy</name>
        <property name="autoload" value="true" />
        <property name="path" value="file:/home/andrew/JcrGroovyTest.groovy" />
      </properties-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

The first is value-param sets JCR repository, the second is value-param sets workspace and the third one is sets JCR node where scripts from plugin will be stored. If specified node does not exist, then it will be created. List of scripts is set by properties-params. Name of each properties-param will be used as node name for stored script, property autoload says to deploy this script at startup time, property path sets the source of script to be loaded. In this example we try to load single script from local file /home/andrew/JcrGroovyTest.groovy.

Load script via HTTP

This is samples of HTTP requests. In this example, we will upload script from file with name test.groovy.

```
andrew@ossl:~> curl -u root:exo \
-X POST \
-H 'Content-type:script/groovy' \
--data-binary @test.groovy \
http://localhost:8080/rest/script/groovy/add/repository/production/script/groovy/test.groovy
```

This example imitate sending data with HTML form ('multipart/form-data'). Parameter autoload is optional. If parameter autoload=true then script will be instantiate and deploy script immediately.

```
andrew@ossl:~> curl -u root:exo \
-X POST \
-F "file=@test.groovy;name=test" \
-F "autoload=true" \
```

```
http://localhost:8080/rest/script/groovy/add/repository/production/script/groovy/test1.groovy
```

Instantiation

org.exoplatform.services.script.groovy.GroovyScriptInstantiator is part of project exo.core.component.script.groovy. GroovyScriptInstantiator can load script from specified URL and parse stream that contains Groovy source code. It has possibility inject component from Container in Groovy Class constructor. Configuration example:

```
<component>
  <type>org.exoplatform.services.script.groovy.GroovyScriptInstantiator</type>
</component>
```

Deploying newly created Class as RESTful service

To deploy script automatically at server statup time, its property exo:autoload must be set as true. org.exoplatform.services.jcr.ext.script.groovy.GroovyScript2RestLoader check JCR workspaces which were specified in configuration and deploy all auto-loadable scripts.

Example of configuration.

```
<component>
    <type>org.exoplatform.services.jcr.ext.script.groovy.GroovyScript2RestLoader</type>
    <init-params>
      <object-param>
        <name>observation.config</name>
                                                                                <object
 type="org.exoplatform.services.jcr.ext.script.groovy.ObservationListenerConfiguration">
          <field name="repository">
            <string>repository</string>
          </field>
          <field name="workspaces">
            <collection type="java.util.ArrayList">
              <value>
                <string>production</string>
              </value>
            </collection>
          </field>
        </object>
      </object-param>
    </init-params>
```

```
  </component>
```

In example above JCR workspace "production" will be checked for autoload scripts. At once, this workspace will be listened for changes script's source code (property jcr:data).

Script Lifecycle Management

If GroovyScript2RestLoader configured as was decribed in the previous section, then all "autoload" scripts deployed. In the first section, we added script from file /home/andrew/JcrGroovyTest.groovy to JCR node /script/groovy/JcrGroovyTest.groovy, repository repository, workspace production. In section "Load script via HTTP", it was refered about load scripts via HTTP, there is an opportunity to manage the life cycle of script.

Undeploy script, which is alredy deployed:

```
andrew@ossl:~> curl -u root:exo \
-X GET \
http://localhost:8080/rest/script/groovy/load/repository/production/script/groovy/
JcrGroovyTest.groovy?state=false
```

Then deploy it again:

```
andrew@ossl:~> curl -u root:exo \
-X GET \
http://localhost:8080/rest/script/groovy/load/repository/production/script/groovy/
JcrGroovyTest.groovy?state=true
```

or even more simple:

```
andrew@ossl:~> curl -u root:exo \
-X GET \
http://localhost:8080/rest/script/groovy/load/repository/production/script/groovy/
JcrGroovyTest.groovy
```

Disable scripts autoloading, NOTE it does not change current state:

```
andrew@ossl:~> curl -u root:exo \
-X GET \
```

```
http://localhost:8080/rest/script/groovy/repository/production/script/groovy/
JcrGroovyTest.groovy/autoload?state=false
```

Enable it again:

```
andrew@ossl:~> curl -u root:exo \
-X GET \
http://localhost:8080/rest/script/groovy/autoload/repository/production/script/groovy/
JcrGroovyTest.groovy?state=true
```

and again more simpe variant:

```
andrew@ossl:~> curl -u root:exo \
-X GET \
http://localhost:8080/rest/script/groovy/autoload/repository/production/script/groovy/
JcrGroovyTest.groovy
```

Change script source code:

```
andrew@ossl:~> curl -u root:exo \
-X POST \
-H 'Content-type:script/groovy' \
--data-binary @JcrGroovyTest.groovy \
http://localhost:8080/rest/script/groovy/update/repository/production/script/groovy/
JcrGroovyTest.groovy
```

This example imitates sending data with HTML form ('multipart/form-data').

```
andrew@ossl:~> curl -u root:exo \
-X POST \
-F "file=@JcrGroovyTest.groovy;name=test" \
http://localhost:8080/rest/script/groovy/update/repository/production/script/groovy/
JcrGroovyTest.groovy
```

Remove script from JCR:

```
andrew@ossl:~> curl -u root:exo \
-X GET \
```

```
http://localhost:8080/rest/script/groovy/delete/repository/production/script/groovy/
JcrGroovyTest.groovy
```

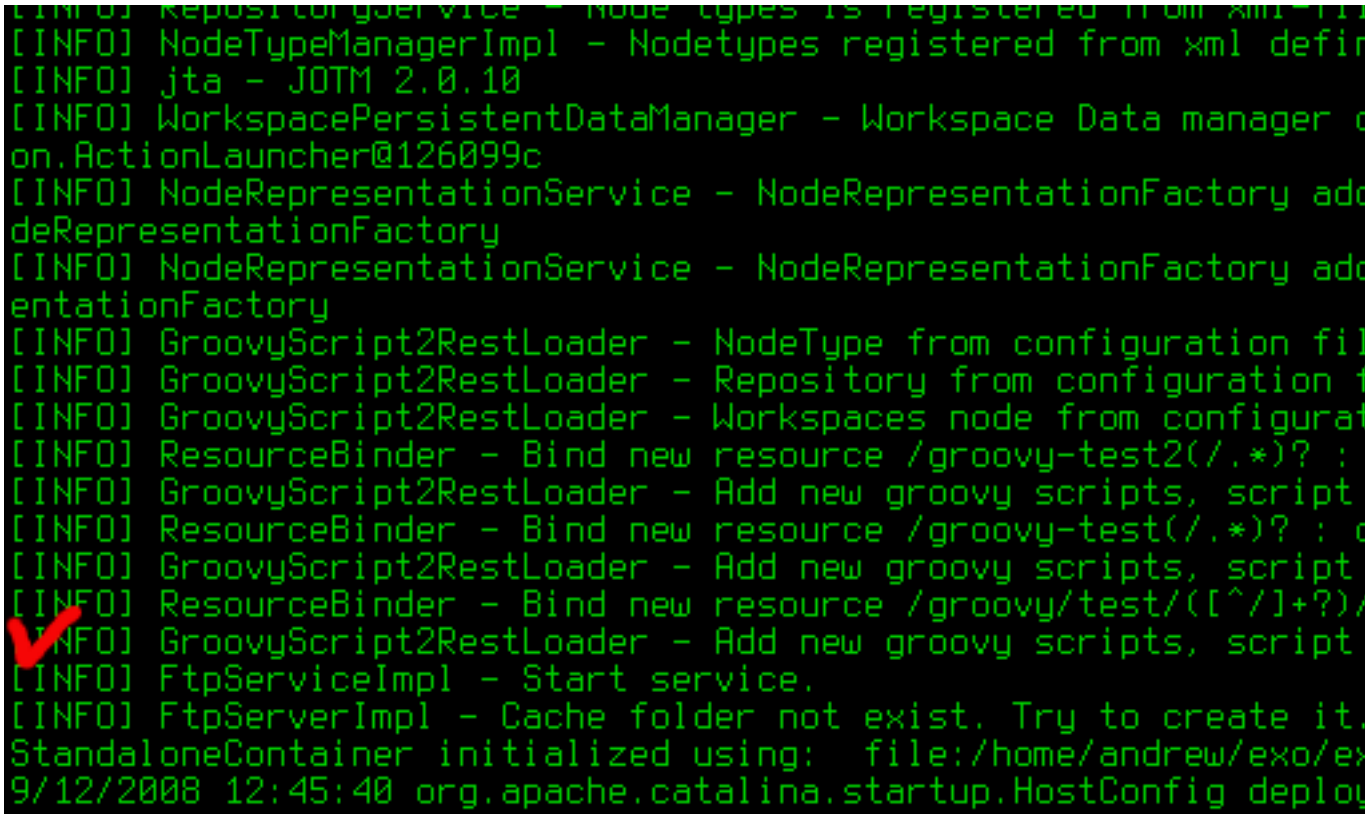
Getting node UUID example

Now we are going to try simple example of Groovy RESTfull service. There is one limitation, even if we use groovy, we should use Java style code and decline to use dynamic types, but of course we can use it in private methods and feilds. Create file JcrGroovyTest.groovy, in this example I save it in my home directory /home/andrew/JcrGroovyTest.groovy. Then, configure GroovyScript2RestLoaderPlugin as described in section Load script at startup time.

```
import javax.jcr.Node
import javax.jcr.Session
import javax.ws.rs.GET
import javax.ws.rs.Path
import javax.ws.rs.PathParam
import org.exoplatform.services.jcr.RepositoryService
import org.exoplatform.services.jcr.ext.app.ThreadLocalSessionProviderService

@Path("groovy/test/{repository}/{workspace}")
public class JcrGroovyTest {
  private RepositoryService                repositoryService
  private ThreadLocalSessionProviderService sessionProviderService

  public JcrGroovyTest(RepositoryService repositoryService,
                       ThreadLocalSessionProviderService sessionProviderService) {
    this.repositoryService = repositoryService
    this.sessionProviderService = sessionProviderService
  }


  @GET
  @Path("{path:.*}")
  public String nodeUUID(@PathParam("repository") String repository,
                         @PathParam("workspace") String workspace,
                         @PathParam("path") String path) {
    Session ses = null
    try {
                    ses    =    sessionProviderService.getSessionProvider(null).getSession(workspace,
 repositoryService.getRepository(repository))
      Node node = (Node) ses.getItem("/" + path)
      return node.getUUID() + "\n"
    } finally {
```

```
    if (ses != null)
      ses.logout()
  }
}
```

After configuration is done, start the server. If configuration is correct and script does not have syntax error, you should see next:



In the screenshot, we can see the service deployed.

First, create a folder via WebDAV in the repository production, folder name 'test'. Now, we can try access this service. Open another console and type command:

```
andrew@ossl:~> curl -u root:exo \
http://localhost:8080/rest/groovy/test/repository/production/test
```

Whe you try to execute this command, you should have exception, because JCR node '/test' is not referenceable and has not UUID. We can try add mixin mix:referenceable. To do this, add one more method in script. Open script from local source code /home/andrew/JcrGroovyTest.groovy, add following code and save file.

```
@POST
```

```
@Path("{path:.*}")
public void addReferenceableMixin(@PathParam("repository") String repository,
                                  @PathParam("workspace") String workspace,
                                  @PathParam("path") String path) {
  Session ses = null
  try {
                ses     =     sessionProviderService.getSessionProvider(null).getSession(workspace,
 repositoryService.getRepository(repository))
    Node node = (Node) ses.getItem("/" + path)
    node.addMixin("mix:referenceable")
    ses.save()
  } finally {
    if (ses != null)
      ses.logout()
  }
}
```

Now we can try to change script deployed on the server without server restart. Type in console next command:

```
andrew@ossl:~> curl -i -v -u root:exo \
-X POST \
--data-binary @JcrGroovyTest.groovy \
-H 'Content-type:script/groovy' \
http://localhost:8080/rest/script/groovy/update/repository/production/script/groovy/
JcrGroovyTest.groovy
```

Node '/script/groovy/JcrGroovyTest.groovy' has property exo:autoload=true so script will be re-deployed automatically when script source code changed.

Script was redeployed, now try to access a newly created method.

```
andrew@ossl:~> curl -u root:exo \
-X POST \
http://localhost:8080/rest/groovy/test/repository/production/test
```

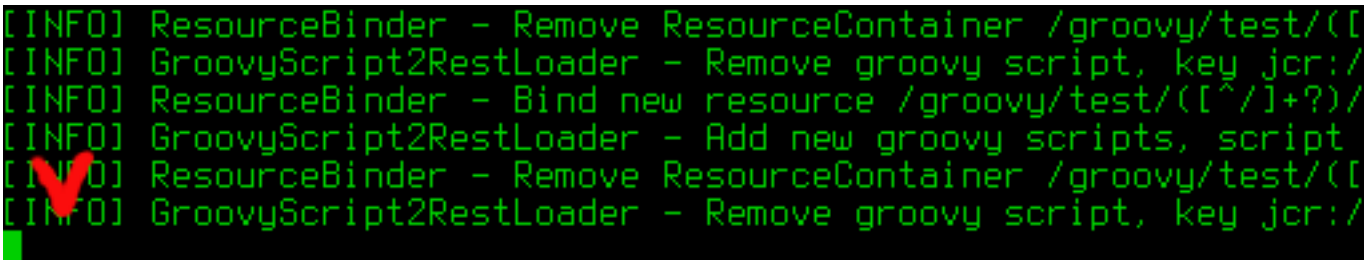
Method excution should be quiet, without output, traces, etc. Then we can try again get node UUID.

```
andrew@ossl:~> curl -u root:exo \
http://localhost:8080/rest/groovy/test/repository/production/test
1b8c88d37f0000020084433d3af4941f
```

Node UUID: 1b8c88d37f0000020084433d3af4941f

We don't need this scripts any more, so remove it from JCR.

```
andrew@ossl:~> curl -u root:exo \
http://localhost:8080/rest/script/groovy/delete/repository/production/script/groovy/
JcrGroovyTest.groovy
```



Groovy script restrictions

You should keep one class per one groovy file. The same actually for interface and it implementation. It's limitation of groovy parser that does not have type Class[] parseClass(InputStream) or Collection parseClass(InputStream) but only Class parseClass(InputStream) instead.

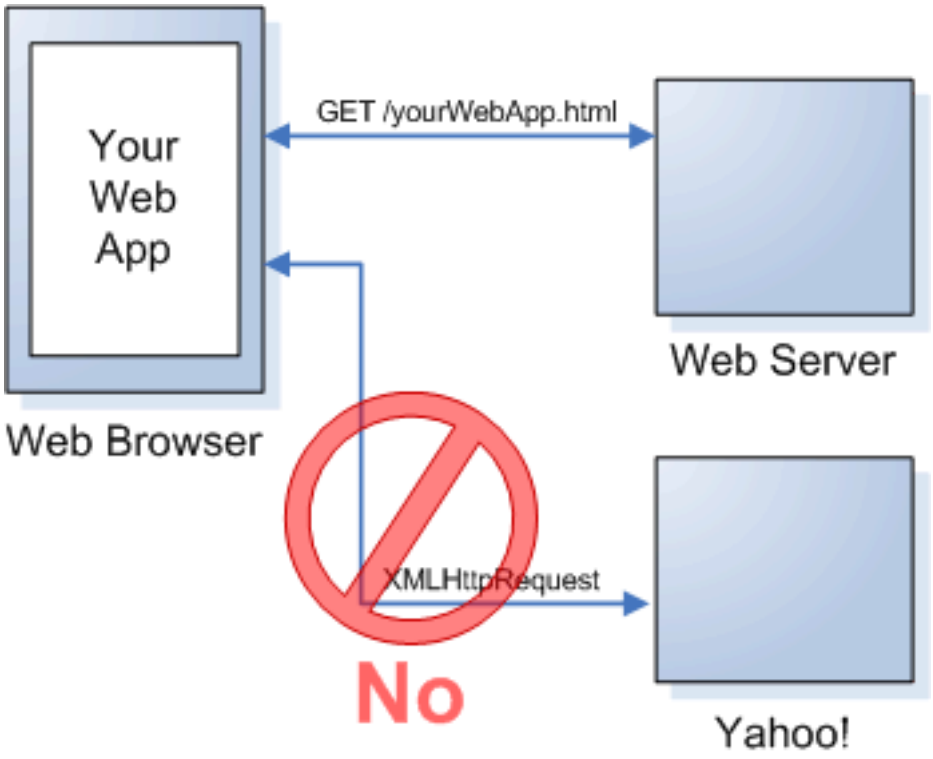
That is all.

Framework for cross-domain AJAX

(*https://anonsvn.jboss.org/repos/exo-jcr/ws/trunk/exo.ws.frameworks.javascript.cross-domain-ajax*)

Motivation

XmlHttpRequest objects are bound by the same origin security policy of browsers, which prevents a page from accessing data from another server. This has put a serious limitation on Ajax developers: you can use XmlHttpRequests to make background calls to a server, but it has to be the same server that served up the current page. For more details, you can visit *http://www.mozilla.org/projects/security/components/same-origin.html*.



But actually writing client web applications that use this object can be tricky given restrictions imposed by web browsers on network connections across domains. So you need to find the way to bypass this limitation of AJAX.

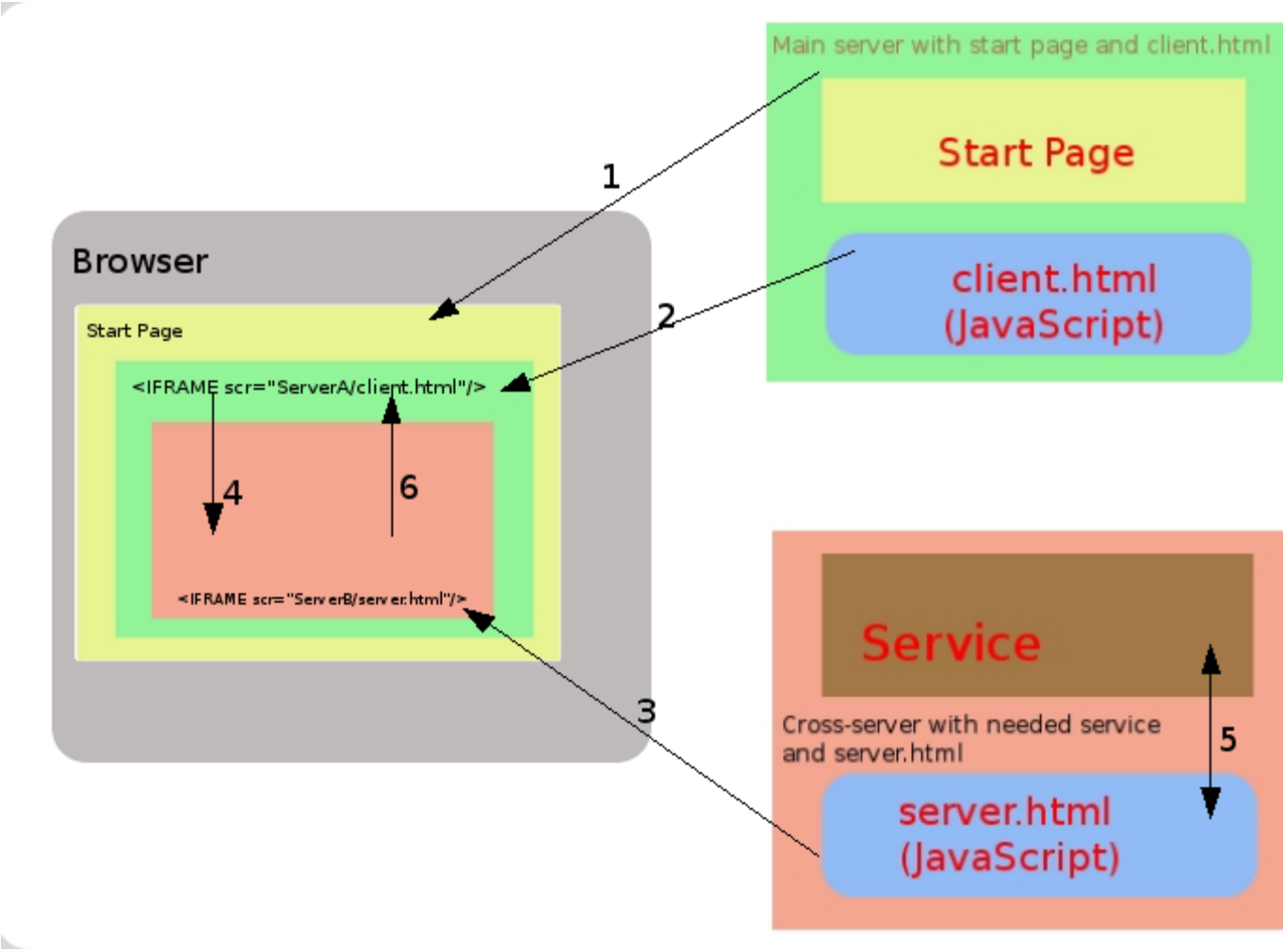
Scheme (how it works)

To describe our method for cross-domain AJAX solution, let's consider the following scheme contains of 3 components:

1). User agent (a browser).

2). ServerA contains a main page with dedicated client and server IFRAMEs (see below) and an HTML client page (client.html) referenced from the client IFRAME. This client page contains dedicated script to push the data for request into server IFRAME.

3). ServerB contains remote service that want get access to and an HTML server page (server.html) referenced from the server IFRAME. This server page contains dedicated script to push the requested data into client IFRAME.



A Working Sequence:

1) A Browser requests the Start page from the ServerA

2) The Start page is retrieved from the ServerA.

3) Create in the start page IFRAME (name it - "client iframe") and insert it in the document from ServerA (client.

4) In "client iframe" create ne IFRAME element ("server iframe") and insert it in the document from ServerB (server.html). Documents (client.html and server.html) contain special script that can transfer data between ifarmes.

5) "Client iframe" transfer information about HTTP method and URL that we want do cross-domain request to "server iframe".

6) "Server iframe" do simple XmlHttpRequest to the service that we need (it can do that because download from same domain) and get informaton from service.

7) "Server iframe" transfer data to "client iframe" and now we get information that we want.

How to use it

1). Place the file client.html and xda.js on the serverA.

2). Place the file server.html on the serverB.

3). Declare xda.js in the main page.

```
<script type="text/javascript" src="xda.js"></script>
```

4). Create JS function which performs cross domain call as in the following example:

```
<script type="text/javascript">

 function test(){
            var facade = xdaInit();
                facade.clientURI = "http://localhost/cross-domain-ajax/client/client.html";

                facade.serverURI = "http://localhost:8080/cross-domain-ajax/server/server.html";

                facade.apiURI = "http://localhost:8080/cross-domain-ajax/server/test.txt";

            facade.method = "POST";
            facade.load = function(result) {
                        alert(result.responseText);
              }
                facade.setRequestHeader("keep-alive","200");

            xda.create(facade);
       }
</script>
```

5). Use this function (here it is bound to a button's onclick event).

```
<button onclick='test()'>test cross-domain</button>
```

Part V. Frequently Asked Question

JCR FAQ

It's the draft for a future FAQ of JCR usage.

Kernel

What is the best, standardized way to get the instance of a service ?

```
container.getComponentInstanceOfType(ServiceName.class);
```

JCR

JCR core

Is it better to use Session.getNodeByUUID or Session.getItem?

Session.getNodeByUUID() about 2.5 times faster of Session.getItem(String) and only 25% faster of Node.getNode(String). See the daily tests results for such comparisons, e.g.

http://tests.exoplatform.org/JCR/1.12.2-GA/rev.2442/daily-performance-testing-results/jcr.core./index.html

Does it make sense to have all the node referencable to use getNodeByUUID all the time?

Until it's applicable for a business logic it can be. But take in account the paths are human readable and lets you think in hierarchy. If it's important a location based approach is preferable.

What should I use to check if an Item exists before getting the Value?

Use Session.itemExists(String absPath), Node.hasNode(String relPath) or Property.hasProperty(String name). It's also is possible to check Node.hasNodes() and Node.hasProprties().

How to use Observation properly?

JCR Observation's a way to listen on persistence changes of a Repository. It provides several options to configure the listener for an interesting only changes. To use properly, it's important to understand concept of events filtering for a registered EventListener (8.3.3 Observation Manager). An often confusing part, it's the **absPath**, it's an associated parent of a location you want to observe events on. I.e. it's a parent of child node(s) or this parent property(ies); if **isDeep** is true then you'll

get events of all the subtree of child nodes also. The same actual for **uuid** and **nodeTypeName** parameters of ObservationManager.addEventListener() method.

Is it better to use queries that to access the data by the JCR API?

No, direct access to items via JCR API is more efficient. Search will consume additional resources for index querying and only then return the items.

What is default query ordering?

By default (if query do not contains any ordering statements) result nodes is sorted by document order.

Is ordering by jcr:path or Item name supported?

No, it does not supported. There is two ways to ordering results, when path may be used as criteria:

- Order by property with value type NAME or PATH (jcr supports it).
- Order by jcr:path - sort by exact path of node (jcr do not supports it).

Order by jcr:path

If no order specification is supplied in the query statement, implementations may support document order on the result nodes (see 6.6.4.2 Document Order). And it is sorted by order number.

By default, (if query do not contains any ordering statements) result nodes is sorted by document order.

```
SELECT * FROM nt:unstructured WHERE jcr:path LIKE 'testRoot/%'
```

For specified jcr:path ordering there is different proceeding in XPath and SQL:

- SQL no matter ascending or descending - query returns result nodes in random order: {code}SELECT * FROM nt:unstructured WHERE jcr:path LIKE 'testRoot/%' ORDER BY jcr:path{code}
- XPath - jcr:path order construction is ignored (so result is not sorted according path); {code}/testRoot/* *@jcr:primaryType='nt:unstructured'* order by jcr:path{code}

How eXo JCR indexer uses content encoding?

1. Indexer uses jcr:encoding property of nt:resource node (used as jcr:content child node of nt:file) 2. if no jcr:encoding property set the Document Service will use the one configured in the service (defaultEncoding) 3. if nothing is configured a JVM, the default encoding will be used

Which database server is better for eXo JCR?

If the question is a performance, it's difficult question, as each database can be configured to be more (and more) faster for each special case. MySQL with MyISAM engine will be faster. But MySQL has limitations for indexes for multilingual columns (Item names actually). So, with long Item names (larger ofOracle or PostgreSQL also are good for performance. DB2 and MSSQL are slower in default configurations. Default configuration of Sybase leader of slowness. But in this question, take the database server maintenance in account. MySQL and PostgreSQL are simple in installation and can work even on limited hardware. Oracle, DB2, MSSQL or Sybase need more efforts. The same actual for maintenance during the work. Note for Sybase: "check-sns-new-connection" data container configuration parameter should be set to "true". For testing purpose, embedded database such as HSQLDB is the best choice. Apache Derby and H2 also supported. But H2 surprisingly needs "beta" feature enabled - MVCC=TRUE in JDBC url.

How to setup eXo JCR for mutilingial content on MySQL?

To allow multiple character sets to be sent from the client, the UTF-8 encoding should be used, either by configuring utf8 as the default server character set, or by configuring the JDBC driver to use UTF-8 through the characterEncoding property. MySQL database should be created in single-byte encoding, e.g. "latin1":

```
CREATE DATABASE db1 CHARACTER SET latin1 COLLATE latin1_general_cs;
```

eXo JCR application (e.g. GateIn) should use JCR dialect "MySQL-UTF8".

In other words: MySQL database default encoding and JCR dialect cannot be UTF8 both. Use single-byte encoding (e.g. "latin1") for database and "mysql-utf8" dialect for eXo JCR.

Notice: "MySQL-UTF8" dialect cannot be auto-detected, it should be set explicitly in configuration.

Does MySQL have limitation affecting on eXo JCR features?

Index's key length of JCR_SITEM (JCR_MITEM) table for mysql-utf8 dialect is reduced to 765 bytes (or 255 chars).

Does use of Sybase database need special options in eXo JCR configuration?

To enable JCR working properly with Sybase, a property 'check-sns-new-connection' with 'false' value is required for each workspace data container:

```
<container
 class="org.exoplatform.services.jcr.impl.storage.jdbc.optimisation.CQJDBCWorkspaceDataContainer">
  <properties>
    <property name="source-name" value="jdbcjcr" />
```

```
    <property name="dialect" value="auto" />
    <property name="multi-db" value="true" />
    <property name="update-storage" value="false" />
    <property name="max-buffer-size" value="200k" />
    <property name="swap-directory" value="target/temp/swap/ws" />
    <property name="swap-directory" value="target/temp/swap/ws" />
    <property name="check-sns-new-connection" value="false" />
  </properties>
```

How to open and close a session properly to avoid memory leaks?

```
Session session = repository.login(credentials);
try
{
// here your code
}
finally
{
session.logout();
}
```

Can I use Session after loging out?

No. Any instance of Session or Node (acquired through session) shouldn't be used after loging out anymore. At least, it is highly recommended not to use.

How to configure jcr for cluster ?

So we have configured JCR in standalone mode and want to reconfigure it for clustered environment. First of all, let's check whether all requirements are satisfied:

- Dedicated RDBMS anyone like MySQL, Postges, Oracle and, etc but just not HSSQL;
- Shared storage. The simplest thing is to use shared FS like NFS or SMB mounted in operation system, but they are rather slow. The best thing is to use SAN (Storage Area Network);
- Fast network between JCR nodes.

So now, need to configure Container a bit. Check exo-configuration.xml to be sure that you are using JBossTS Transaction Service and JBossCache Transaction Manager, as shown below.

```
<component>
  <key>org.jboss.cache.transaction.TransactionManagerLookup</key>
```

```
  <type>org.jboss.cache.GenericTransactionManagerLookup</type>
</component>

<component>
  <key>org.exoplatform.services.transaction.TransactionService</key>
  <type>org.exoplatform.services.transaction.jbosscache.JBossTransactionsService</type>
  <init-params>
    <value-param>
      <name>timeout</name>
      <value>300</value>
    </value-param>
  </init-params>
</component>
```

Next stage is actually the JCR configuration. We need JBossCache configuration templates for : data-cache, indexer-cache and lock-manager-cache. Later they will be used to configure JCR's core components. There are pre-bundled templates in EAR or JAR in conf/standalone/cluster. They can be used as is or re-written if needed. And now, re-configure a bit each workspace. Actually, a few parameters need changing, e.g. <cache>, <query-handler> and <lock-manager>.

- <cache> configuration should look like this:

```
<cache enabled="true"

 class="org.exoplatform.services.jcr.impl.dataflow.persistent.jbosscache.JBossCacheWorkspaceStorageCache">
    <properties>
      <property name="jbosscache-configuration" value="test-jbosscache-data.xml" />
      <property name="jgroups-configuration" value="udp-mux.xml" />
      <property name="jgroups-multiplexer-stack" value="true" />
      <property name="jbosscache-cluster-name" value="JCR-cluster-db1-ws" />
    </properties>
</cache>
```

  - "jbosscache-configuration" is the path to configuration template;
  - "jgroups-configuration" is path to JGroups configuration that is multiplexer stack is used (default). This file is also pre-bundled with templates and is recommended for use;
  - "jgroups-multiplexer-stack" just simply "true". Strongly recommended;
  - "jbosscache-cluster-name" is the name of cluster group. It should be different for each workspace and each workspace component. I.e.: <repository_name>-<ws_name>-<component(cache\|lock\|index)>

- <query-handler> configuration
  - You must replace or add in <query-handler> block the "changesfilter-class" parameter equals with:

```
<property                                                    name="changesfilter-class"

>
```

  - add JBossCache-oriented configuration:

```
<property name="jbosscache-configuration" value="test-jbosscache-indexer.xml" />
<property name="jgroups-configuration" value="udp-mux.xml" />
<property name="jgroups-multiplexer-stack" value="true" />
<property name="jbosscache-cluster-name" value="JCR-cluster-indexer-db1-ws" />
<property name="max-volatile-time" value="60" />
```

Those properties have the same meaning and restrictions as in the previous block. The last property "max-volatile-time" is not mandatory but recommended. This notifies that the latest changes in index will be visible for each cluster node not later than in 60s.

- <lock-manager> configuration

  Maybe this is the hardest element to configure, because we have to define access to DB where locks will be stored. Replace exsiting lock-manager configuration with shown below.

```
<lock-manager
 class="org.exoplatform.services.jcr.impl.core.lock.jbosscache.CacheableLockManagerImpl">
  <properties>
    <property name="time-out" value="15m" />
    <property name="jbosscache-configuration" value="test-jbosscache-lock.xml" />
    <property name="jgroups-configuration" value="udp-mux.xml" />
    <property name="jgroups-multiplexer-stack" value="true" />
    <property name="jbosscache-cluster-name" value="JCR-cluster-locks-db1-ws" />
    <property name="jbosscache-cl-cache.jdbc.table.name" value="jcrlocks_db1_ws" />
    <property name="jbosscache-cl-cache.jdbc.table.create" value="true" />
    <property name="jbosscache-cl-cache.jdbc.table.drop" value="false" />
   <property name="jbosscache-cl-cache.jdbc.table.primarykey" value="jcrlocks_db1_ws_pk"
 />
    <property name="jbosscache-cl-cache.jdbc.fqn.column" value="fqn" />
    <property name="jbosscache-cl-cache.jdbc.node.column" value="node" />
```

```
    <property name="jbosscache-cl-cache.jdbc.parent.column" value="parent" />
    <property name="jbosscache-cl-cache.jdbc.datasource" value="jdbcjcr" />
  </properties>
</lock-manager>
```

First few properties are the same as in the previous components, but here you can see some strange "jbosscache-cl-cache.jdbc.*" properties. They define access parameters for database where lock are persisted.

- "jbosscache-cl-cache.jdbc.table.create" - whether to create it or not. Usually "true";
- "jbosscache-cl-cache.jdbc.table.drop" - whether to drop on a start or not. Use "false";
- "jbosscache-cl-cache.jdbc.table.primarykey" - the name of column with pk;
- "jbosscache-cl-cache.jdbc.fqn.column" - the name of one more column. If you are not sure how to use, follow the example above (if much interested, please refer to JBossCache JDBCCacheLoader documentation)
- "jbosscache-cl-cache.jdbc.node.column" - the name of one more column. If you are not sure how to use, follow the example above (if much interested, please refer to JBossCache JDBCCacheLoader documentation)
- "jbosscache-cl-cache.jdbc.parent.column" - name of one more column. If you are not sure how to use, follow the example above if you are not sure (if much interested, please refer to JBossCache JDBCCacheLoader documentation)
- "jbosscache-cl-cache.jdbc.datasource" - name of configured in Container datasource, where you want to store locks. The best idea is to use the same as for workspace.

That's all. JCR is ready for running in a cluster.

How to use lucene spellchecker?

There is few steps:

- Enable lucene spellchecker in jcr QueryHandler configuration:

```
<query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
  <properties>
    ...
                                          <property          name="spellchecker-class"

 />
    ...
```

```
    </properties>
  </query-handler>
```

- Execute query with rep:spellcheck function and word that is checked:

```
Query query = qm.createQuery("select rep:spellcheck() from nt:base where " + "jcr:path = '/'
 and spellcheck('word that is checked')", Query.SQL);
RowIterator rows = query.execute().getRows();
```

- Fetch a result:

```
Row r = rows.nextRow();
Value v = r.getValue("rep:spellcheck()");
```

If there is no any results, that means there is no suggestion, so word is correct or spellcheckers dictionary do not contain any words like the checked word.

How can I affect to spellchecker results?

There is two parameters in jcr QueryHandler configuration:

- Minimal distance between checked word and proposed suggestion;
- Search for more popular suggestions;

```
<query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
  <properties>
    ...
                                                <property          name="spellchecker-class"

 />
    <property name="spellchecker-more-popular" value="false" />
    <property name="spellchecker-min-distance" value="0.55" />
    ...
  </properties>
</query-handler>
```

Minimal distance is counted as Levenshtein distance between checked word and spellchecker suggestion.

MorePopular paramter affects in next way: If "morePopular" disabled:

- If the proposed word exists in the directory - no suggestion given;
- If the proposed word doesn't exist in the directory - propose the closed word;

If "morePopular" enabled:

- No matter word exists or not, checker will propose the closed word that is more popular than the checked word.

JCR extensions

How to restore repository to existing repository ?

1. Remove existing repository, use:

```
RepositoryService.removeRepository(String repositoryName)
```

2. Restore repository, use

```
BackupManager.restore(RepositoryBackupChainLog log, RepositoryEntry repositoryEntry, boolean asynchronous)
```

How to restore workspace to existing worksapce?

1. Remove existing workspace, use:

```
ManageableRepository.removeWorkspace(String workspaceName)
```

2. Restore workspace, use:

```
BackupManager.restore(BackupChainLog log, String repositoryName, WorkspaceEntry workspaceEntry, boolean asynchronous)
```

Does JCR support hot backup?

Yes, JCR is support hot backup. Will use org.exoplatform.services.jcr.ext.backup.BackupManager.

WebDAV

I uploaded a file to WebDAV server using Mac OS Finder, but the file size is '0', what is wrong ?

This is known as a finder bug started from Mac OS v.10.5.3 and not yet fixed, .

For more details follow: *Apple Disscussion thread.* [http://discussions.apple.com/thread.jspa?threadID=1538882&start=0&tstart=0]

Can I manage 'cache-control' value for different media-types from server configuration ?

Use "cache-control" configuration parameter.

The value of this parameter must contain colon-separated pairs "MediaType:cache-control value"

For example, if you need to cache all text/xml and text/plain files for 5 minutes (300 sec.) and other text/* files for 10 minutes (600 sec.), use the next configuration:

```
<component>
  <type>org.exoplatform.services.jcr.webdav.WebDavServiceImpl</type>
  <init-params>
    <value-param>
      <name>cache-control</name>
      <value>text/xml,text/plain:max-age=300;text/*:max-age=600;</value>
    </value-param>
  <init-params>
<component>
```

How to perform WebDAV requests using curl ?

Simple Requests

For simple request such as: GET, HEAD, MKCOL, COPY, MOVE, DELETE, CHECKIN, CHECKOUT, UNCHECKOUT, LOCK, UNLOCK, VERSIONCONTROL, OPTIONS

perform:

```
curl -i -u 'user:pass' -X 'METHOD_NAME' 'resource_url'
```

for example to create a folder named test perform:

```
curl -i -u 'root:exo' -X MKCOL 'http://localhost:8080/rest/jcr/repository/production/test
```

to PUT a test.txt file from your current folder to "test "folder on server perform:

```
curl -i -u 'root:exo' -X PUT 'http://localhost:8080/rest/jcr/repository/production/test/test.txt' -d @test.txt
```

Requests with XML body

For requests which contains xml body such as: ORDER, PROPFIND, PROPPATCH, REPORT, SEARCH

add **-d 'xml_body text'** or **-d @body.xml**

(body.xml must contain a valid xml request bidy.) to you curl-command:

```
curl -i -u 'user:pass'  -X 'METHOD_NAME' -H 'Headers' 'resource_url' -d 'xml_body text'
```

For example about finding all files containing "test" perform:

```
curl -i -u "root:exo" -X "SEARCH" "http://192.168.0.7:8080/rest/jcr/repository/production/" -d
"<?xml version='1.0' encoding='UTF-8' ?>
  <D:searchrequest xmlns:D='DAV:'>
    <D:sql>SELECT * FROM nt:base WHERE contains(*, 'text')</D:sql>
</D:searchrequest>"
```

If you need to add some headers to your request, use \-H key.

To have more information about methods parameters, you can find in *HTTP Extensions for Distributed Authoring* [http://www.ietf.org/rfc/rfc2518.txt] specification.

How eXo JCR WebDAV server treats content encoding?

OS client (Windows, Linux etc) doesn't set an encoding in a request. But eXo JCR WebDAV server looks for an encoding in a Content-Type header and set it to jcr:encoding. See *http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html,* 14.17 Content-Type. e.g. Content-Type: text/html; charset=ISO-8859-4 So, if a client will set Content-Type header, e.g. JS code from a page, it will works for a text file as expected.

If WebDAV request doesn't contain a content encoding, it's possible to write a dedicated action in a customer application. The action will set jcr:encoding using its own logic, e.g. based on IP or user preferences.

Part VI. eXo JCR with GateIn

How to extend my GateIn instance?

Introduction

Overview

Since GateIn beta 2, we added a set of features in order to customize a GateIn instance without modifying the GateIn binary, this usecase will be called *portal extension* in this documentation. Those features are also required to be able to launch several portal instances at the same time, in "eXo terminology" that means to have several "portal.war".

Motivations

Up to now, to create an application over an eXo portal such as DMS, WCM, CS and KS, we need to modify files into the "portal.war". This has many painful consequences, such as:

- It is quite hard to manage from the support point of view since we never know if or how the customer changed his eXo product.
- It is hard to be able to package several eXo products (WCM, CS...) as we need to merge everything manually which is quite error prone.

Finally, at the very beginning, eXo was developed to be able to support several portal instances (which also means several portal containers) but with the time several bad practices made it impossible. So it was important to review the whole code base in order to help/enforce all the GateIn developers to follow the "good practices".

Prerequisites

To be able to migrate an application to GateIn, the first thing we need to do is to ensure that our application supports properly several portal container instances. The following section aims to help you to be compatible with GateIn.

Removing all the hard coded portal container name (i.e. "portal")

Now if we need to get the portal container name (even in a standalone mode: in case of standalone mode the default value will be returned), we can:

- If the component is instantiated by Pico container, you can add in the constructor of your component, the component *ExoContainerContext*, then call the method *getPortalContainerName()*
- If the component is not instantiated by Pico container, you can call at runtime the static method *PortalContainer.getCurrentPortalContainerName()*

- In js files, you can use the variable *currentContext* if your script must be loaded before the variable *eXo.env.server.context*, otherwise use *eXo.env.server.context* instead.
- In jsp files, you can use *request.getContextPath()*.

Removing all the hard coded rest context name (i.e. "rest")

Now if we need to get the rest context name (even in a standalone mode: in case of standalone mode the default value will be returned), we can:

- If the component is instantiated by Pico container, you can add in the constructor of your component, the component *ExoContainerContext*, then call the method *getRestContextName()*
- If the component is not instantiated by Pico container, you can call at runtime the static method *PortalContainer.getCurrentRestContextName()*

Removing all the hard coded realm name (i.e. "exo-domain")

Now if we need to get the realm name (even in a standalone mode: in case of standalone mode the default value will be returned), we can:

- If the component is instantiated by Pico container, you can add in the constructor of your component, the component *ExoContainerContext*, then call the method *getRealmName()*
- If the component is not instantiated by Pico container, you can call at runtime the static method *PortalContainer.getCurrentRealmName()*

Making your Http Filters compatible

Now all your Http Filters that need to get the current *ExoContainer* must extends *org.exoplatform.container.web.AbstractFilter*. You just need to call the method *getContainer()* to get the current *ExoContainer*.

Making your HttpServlets compatible

Now all your HttpServlets that need to get the current *ExoContainer* must extends *org.exoplatform.container.web.AbstractHttpServlet*. This abstract class will ensure that the environment has been properly set, so you will be able to call the usual methods such as *ExoContainerContext.getCurrentContainer()* (if it must also be compatible with the standalone mode) or *PortalContainer.getInstance()* (if it will only work on a portal environment mode).

If you had to implement the method *service(HttpServletRequest req, HttpServletResponse res)*, now you will need to implement *onService(ExoContainer container, HttpServletRequest req, HttpServletResponse res)*, this method will directly give you the current *ExoContainer* in its signature.

In the class *org.exoplatform.container.web.AbstractHttpServlet* you have a method called *requirePortalEnvironment()* that is used to indicate that we would like the abstract class to setup or not the full portal environment (*PortalContainer*, *ClassLoader* and *ServletContext*) before executing the servlet. This value should return true when the servlet is executed within the web application of a portal container. By default, it checks if the name of the current *ServletContext* is a portal container name, it is sufficient in most of cases but you can still overload this method if you already know that the servlet will always been executed within the web application of portal container (i.e. the method always return true) or will never be executed within the web application of a portal container (i.e. the method always return false) .

Making your HttpSessionListeners compatible

Now all your HttpSessionListeners that need to get the current *ExoContainer* must extends *org.exoplatform.container.web.AbstractHttpSessionListener*. This abstract class will give you the current *ExoContainer* directly in the signature of the methods to implement which are _ onSessionCreated(ExoContainer container, HttpSessionEvent event)_ and *onSessionDestroyed(ExoContainer container, HttpSessionEvent event)*

You will also need to implement the method called *requirePortalEnvironment()* that is used to indicate that we would like the abstract class to setup or not the full portal environment (*PortalContainer* and *ClassLoader*) before processing the event. This value should return true when the event is processed within the web application of a portal container.

Use init tasks if you need a PortalContainer to initialize an Http Filter or an HttpServlet

If your Http *Filter* or your *HttpServlet* requires a PortalContainer to initialize, you need to convert your code in order to launch the code responsible for the initialization in the method *onAlreadyExists* of an *org.exoplatform.container.RootContainer.PortalContainerInitTask*.

We need to rely on init tasks, in order to be sure that the portal container is at the right state when the task is executed, in other words the task could be delayed if you try to execute it too early. Each task is linked to a web application, so when we add a new task, we first retrieve all the portal containers that depend on this web application according to the *PortalContainerDefinitions*, and for each container we add the task in a sorted queue which order is in fact the order of the web applications dependencies defined in the *PortalContainerDefinition*. If no *PortalContainerDefinition* can be found we execute synchronously the task which is in fact the old behavior (i.e. without the starter).

The supported init tasks are:

- The *org.exoplatform.container.RootContainer.PortalContainerPreInitTask* which are executed before the portal container has been initialized

- The *org.exoplatform.container.RootContainer.PortalContainerPostInitTask* which are executed after the portal container has been initialized
- The *org.exoplatform.container.RootContainer.PortalContainerPostCreateTask* which are executed after the portal container has been fully created (i.e. after all the post init tasks).

An init task is defined as below:

PortalContainerPreInitTask

```
/**
 * This interface is used to define a task that needs to be launched at a given state during the
 * initialization of a portal container
 */
public static interface PortalContainerInitTask
{

  /**
   * This method allows the implementation to define what the state "already exists"
   * means for a portal container
   *
   * @param portalContainer the value of the current portal container
   * @return <code>true</code> if the portal container exists according to the task
   * requirements, <code>false</code> otherwise
   */
  public boolean alreadyExists(PortalContainer portalContainer);

  /**
   * This method is called if the related portal container has already been registered
   *
   * @param context the servlet context of the web application
   * @param portalContainer the value of the current portal container
   */
  public void onAlreadyExists(ServletContext context, PortalContainer portalContainer);

  /**
   * Executes the task
   *
   * @param context the servlet context of the web application
   * @param container The portal container on which we would like to execute the task
   */
  public void execute(ServletContext context, PortalContainer portalContainer);

  /**
```

```
     * @return the type of the task
     */
    public String getType();
  }
```

To add a task, you can either call:

- *PortalContainer.addInitTask(ServletContext context, PortalContainerInitTask task)* in order to execute the task on all the portal containers that depend on the given *ServletContext* according to the *PortalContainerDefinitions*.
- *PortalContainer.addInitTask(ServletContext context, PortalContainerInitTask task, String portalContainerName)* in order to execute the task on a given portal container.
- *RootContainer.addInitTask(ServletContext context, PortalContainerInitTask task)* in order to execute the task on the portal container which name is the name of the given *ServletContext*.

We will take for example the class *GadgetRegister* that is used to register new google gadgets on a given portal container.

The old code was:

Old GadgetRegister.java

```
...
public class GadgetRegister implements ServletContextListener
{
...
  public void contextInitialized(ServletContextEvent event)
  {
    try
    {
      ExoContainer pcontainer =  ExoContainerContext.getContainerByName("portal") ;
                                                SourceStorage sourceStorage =
 (SourceStorage)pcontainer.getComponentInstanceOfType(SourceStorage.class);
      ...
  }
  ...
}
```

The new code relies on a *org.exoplatform.container.RootContainer.PortalContainerPostInitTask*, as you can see below

New GadgetRegister.java

```
...
public class GadgetRegister implements ServletContextListener {
...
   public void contextInitialized(ServletContextEvent event)
   {
      // Create a new post init task
      final PortalContainerPostInitTask task = new PortalContainerPostInitTask()
      {

         public void execute(ServletContext context, PortalContainer portalContainer)
         {
            contextInitialized(context, portalContainer);
         }
      };
      // Add the init task to all the related portal containers
      PortalContainer.addInitTask(event.getServletContext(), task);
   }

   private void contextInitialized(ServletContext context, PortalContainer pcontainer)
   {
      try
      {
                                                SourceStorage sourceStorage =
 (SourceStorage)pcontainer.getComponentInstanceOfType(SourceStorage.class);
         ...
   }
  ...
}
```

Making your LoginModules compatible

Now all your LoginModules that need to get the current *ExoContainer* must extends *org.exoplatform.services.security.jaas.AbstractLoginModule*. You just need to call the method *getContainer()* to get the current *ExoContainer*.

The class *org.exoplatform.services.security.jaas.AbstractLoginModule* supports 2 login module options which are *portalContainerName* and *realmName*, to allow you to indicate the realm name and the portal container name, if you want to change the default value.

Avoiding *static* modifier on component dependency

A local variable that stores a component dependency must not be static. In other words, when you create a component A that depends on component B, we don't store B in a static variable of A otherwise we cannot have several different instances of A in the same JVM which is not compatible with a multi-portal instance.

Avoid component initialization based on component dependency in the constructor

We will have more and more extensible components (i.e. that can be extended thanks to an external plugin) which means that those components can only be initialized in the *start* method, thus it is not a good practice to initialize a component in its constructor if this initialization uses other components because those components may not be initialized. For example, now the ResourceBundleService is extensible, so if you create a component that depends on the ResourceBundleService and you need the ResourceBundleService to initialize your component, your component will need to be "Startable" and you will have to initialize your component in a *start* method.

FAQ

What has changed since the previous versions?

The main difference with previous versions is the way to package your application, in the previous versions you had to change the content of the file *portal.war* in order to customize the portal. Now we more consider your application as an add-on that you can packaged in another ear/war file. You just need to follow some rules in order to notify the platform that it must take into account your add-on in order to customize the portal.

Among other things, you will have to:

- Indicate the platform how to initialize and manage your application by defining and registering the *PortalContainerDefinition* related to your portal instance.
- Deploy the *starter* ear/war file that is used to create and start all the portals (i.e. portal containers).

> Please take into account, that you need to ensure that the *starter* is launched after all the other ear/war files.

> If you don't need to customize the portal, you don't have to deploy the starter. The old way to deploy an application is still compatible.

What is the main purpose of a *portal extension*?

An *extension* is just a set of files that we use to customize or add new features to a given portal. An extension must be the least intrusive as possible, as we could potentially have several extensions for the same portal. In other words, we are supposed to only add what is missing in the portal and avoid changing or duplicated files that are in the portal.war.

What is the main purpose of the *starter*?

The *sarter* is a web application that has been added to create and start all the portals (i.e. portal containers) at the same time when all the other web applications have already been started. In fact all other web applications can potentially defined several things at startup such as skins, javascripts, google gadgets and configuration files, thus the loading order is important as we can redefine skins or configuration files or a javascript from a web application 1 could depend on another javascript from a web application 2 so if the web application 2 is loaded after the web application 1, we will get errors in the merged javascript file.

If a *PortalContainerDefinition* has been defined, the loading order will be the order that has been used to define the list of dependency. And if you defined a *PortalContainerDefinition* you need to deploy the starter otherwise the loading order will be the default one (i.e. the loading order of the Application Server)

So if you need to customize your portal by adding a new extension and/or a new portal, you need to defined the related *PortalContainerDefinitions* so you need to deploy also the starter. Otherwise, you don't need to define any *PortalContainerDefinition* and to deploy the *starter*.

How a portal and a portal container are related?

Each portal instance has its own portal container which allows the portal to have its own set of components/services. It will ensure the isolation between the different portal instances.

How to define and register a *PortalContainerDefinition*?

A *PortalContainerDefinition* allows you to indicate the platform how it must initialize and manage your portal. In a *PortalContainerDefinition*, you can define a set of properties, such as:

- The name of the portal container
- The name of the context name of the rest web application
- The name of the realm
- The list of all the dependencies of the portal container ordered by priority

You can define and register a *PortalContainerDefinition* thanks to an external plugin that has to be treated at the *RootContainer* level. In other words, your configuration file must be a file

conf/configuration.xml packaged into a jar file or $AS_HOME/exo-conf/configuration.xml (for more details, please have a look to the article *Container Configuration*).

See below an example of configuration file that define and register a PortalContainerDefinition:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd          http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
  <external-component-plugins>
    <!-- The full qualified name of the PortalContainerConfig -->
            <target-component>org.exoplatform.container.definition.PortalContainerConfig</target-
component>
    <component-plugin>
      <!-- The name of the plugin -->
      <name>Add PortalContainer Definitions</name>
        <!-- The name of the method to call on the PortalContainerConfig in order to register the
 PortalContainerDefinitions -->
      <set-method>registerPlugin</set-method>
      <!-- The full qualified name of the PortalContainerDefinitionPlugin -->
      <type>org.exoplatform.container.definition.PortalContainerDefinitionPlugin</type>
      <init-params>
        <object-param>
          <name>portal</name>
          <object type="org.exoplatform.container.definition.PortalContainerDefinition">
            <!-- The name of the portal container -->
            <field name="name"><string>portal</string></field>
            <!-- The name of the context name of the rest web application -->
            <field name="restContextName"><string>rest</string></field>
            <!-- The name of the realm -->
            <field name="realmName"><string>exo-domain</string></field>
            <!-- All the dependencies of the portal container ordered by loading priority -->
            <field name="dependencies">
              <collection type="java.util.ArrayList">
                <value>
                  <string>eXoResources</string>
                </value>
                <value>
                  <string>portal</string>
                </value>
                <value>
                  <string>dashboard</string>
```

```
            </value>
            <value>
              <string>exoadmin</string>
            </value>
            <value>
              <string>eXoGadgets</string>
            </value>
            <value>
              <string>eXoGadgetServer</string>
            </value>
            <value>
              <string>rest</string>
            </value>
            <value>
              <string>web</string>
            </value>
            <value>
              <string>wsrp-producer</string>
            </value>
            <value>
              <string>sample-ext</string>
            </value>
          </collection>
        </field>
      </object>
    </object-param>
  </init-params>
 </component-plugin>
</external-component-plugins>
</configuration>
```

In the previous example, we define a portal container called "portal", which rest context name is "rest", which realm name is "exo-domain" and which dependencies are the web applications "eXoResources", "portal"... The platform will load first "eXoResources", then "portal" and so on.

How the platform interprets the dependency order defined into the PortalContainerDefinition?

The dependency order defined into the *PortalContainerDefinition* is really crucial since it will be interpreted the same way by several components of the platform. All those components, will consider the 1st element in the list is less important than the second element and so on.

So it is currently used to:

- Know loading order of all the dependencies

- If we have several *PortalContainerConfigOwner* (see next section for more details about a PortalContainerConfigOwner)
  - The ServletContext of all the *PortalContainerConfigOwner* will be unified, if we use the unified ServletContext (PortalContainer.getPortalContext()) to get a resource, it will try to get the resource in the ServletContext of the most important *PortalContainerConfigOwner* (i.e. last in the dependency list) and if it cans find it, it will try with the second most important *PortalContainerConfigOwner* and so on.
  - The ClassLoader of all the *PortalContainerConfigOwner* will be unified, if we use the unified ClassLoader (PortalContainer.getPortalClassLoader()) to get a resource, it will try to get the resource in the ClassLoader of the most important *PortalContainerConfigOwner* (i.e. last in the dependency list) and if it cans find it, it will try with the second most important *PortalContainerConfigOwner* and so on.

How to change the ServletContext name, the realm name and/or the rest context name of my portal without using a PortalContainerDefinition?

To do that you need first to change the default values used by a PortalContainer that has not been defined thanks to a *PortalContainerDefinition*. Those default values can be modified thanks to a set of init parameters of the component *PortalContainerConfig*.

The component *PortalContainerConfig* must be registered at the *RootContainer* level. In other words, your configuration file must be a file *conf/configuration.xml* packaged into a jar file or $AS_HOME/exo-conf/configuration.xml (for more details please have a look to the article *Container Configuration*).

In the example below we will rename:

- The portal name "portal" to "myPortal".
- The rest servlet context name "rest" to "myRest".
- The realm name "exo-domain" to "my-exo-domain".

See below an example

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd          http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
  <component>
    <!-- The full qualified name of the PortalContainerConfig -->
```

```
    <type>org.exoplatform.container.definition.PortalContainerConfig</type>
    <init-params>
      <!-- The name of the default portal container -->
      <value-param>
        <name>default.portal.container</name>
        <value>myPortal</value>
      </value-param>
      <!-- The name of the default rest ServletContext -->
      <value-param>
        <name>default.rest.context</name>
        <value>myRest</value>
      </value-param>
      <!-- The name of the default realm -->
      <value-param>
        <name>default.realm.name</name>
        <value>my-exo-domain</value>
      </value-param>
    </init-params>
  </component>
</configuration>
```

Once your configuration is ready, you need to:

- Update the file *WEB-INF/web.xml* of the file "portal.war" by changing the "display-name" (the new value is "myPortal") and the "realm-name" in the "login-config" (the new value is "my-exo-domain").
- If you use JBoss AS: Update the file *WEB-INF/jboss-web.xml* of the file "portal.war" by changing the "security-domain" (the new value is "java:/jaas/my-exo-domain").
- Rename the "portal.war" to "myPortal.war" (or "02portal.war" to "02myPortal.war")
- Update the file *WEB-INF/web.xml* of the file "rest.war" by changing the "display-name" (the new value is "myRest") and the "realm-name" in the "login-config" (the new value is "my-exo-domain").
- If you use JBoss AS: Update the file *WEB-INF/jboss-web.xml* of the file "rest.war" by changing the "security-domain" (the new value is "java:/jaas/my-exo-domain").
- Rename the "rest.war" to "myRest.war"
- If "portal.war" and "rest.war" were embedded into an ear file: Update the file *META-INF/application.xml* of the file "exoplatform.ear" by remaming "02portal.war" to "02myPortal.war", "portal" to "myPortal", "rest.war" to "myRest.war" and "rest" to "myRest".

The end of the process depends on your application server

On JBoss (tested on JBoss 5.1.0.GA)

You need to change the name of the application policy in your file *conf/login-config.xml* (the new name is "my-exo-domain").

On Tomcat (tested on Tomcat 6.0.20)

You need to:

- Update the file *tomcat/conf/Catalina/localhost/portal.xml* by changing the "path" (the new value is "/myPortal"), the "docBase" (the new value is "myPortal") and the "appName" in the "Realm" definition (the new value is "my-exo-domain").
- Rename the file *tomcat/conf/Catalina/localhost/portal.xml* to *myPortal.xml*.
- Update the file *tomcat/conf/Catalina/localhost/rest.xml* by changing the "path" (the new value is "/myRest"), the "docBase" (the new value is "myRest") and the "appName" in the "Realm" definition (the new value is "my-exo-domain").
- Rename the file *tomcat/conf/Catalina/localhost/rest.xml* to *myRest.xml*.
- Change the realm name in the file *tomcat/conf/jaas.conf* (the new name is "my-exo-domain").

How to add new configuration file to a given portal from a war file?

To indicate the platform that a given web application has configuration file to provide, you need to:

- Add the ServletContextListener *org.exoplatform.container.web.PortalContainerConfigOwner* in its web.xml.
- Add the servlet context name of this web application as a new dependency in the *PortalContainerDefinition* of all the portal containers for which you want to share the configuration file embedded into the war file, located at *WEB-INF/conf/configuration.xml*.

The simple fact to add this Servlet Context Listener, will add the Servlet Context of this web application to the Unified Servlet Context of all the *PortalContainers* that depend on this web application according to their *PortalContainerDefinition*.

> The position of the servlet context name of this web application in the dependency list is important since the last configuration file loaded has always right towards other configuration files. Each configuration file loaded, could potentially redefine a configuration file that has already been loaded. Moreover, as we now use a unified Servlet Context to load the configuration files, if you want for instance to import the file *war:/conf/*

database/database-configuration.xml and this file exists in 2 different web applications, the file from the last (according to the dependency order) web application will be loaded.

A portal is implicitly considered as a *PortalContainerConfigOwner* without having to define the ServletContextListener *org.exoplatform.container.web.PortalContainerConfigOwner* in its web.xml.

See an example of a web.xml below:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
                "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>sample-ext</display-name>

  <context-param>
                                                                                <param-
name>org.exoplatform.frameworks.jcr.command.web.fckeditor.digitalAssetsWorkspace</
param-name>
    <param-value>collaboration</param-value>
    <description>Binary assets workspace name</description>
  </context-param>

  <context-param>
       <param-name>org.exoplatform.frameworks.jcr.command.web.fckeditor.digitalAssetsPath</
param-name>
    <param-value>/Digital Assets/</param-value>
    <description>Binary assets path</description>
  </context-param>

  <context-param>
    <param-name>CurrentFolder</param-name>
    <param-value>/Digital Assets/</param-value>
    <description>Binary assets workspace name</description>
  </context-param>

  <!-- ================================================================ -->
  <!--   RESOURCE FILTER TO CACHE MERGED JAVASCRIPT AND CSS             -->
  <!-- ================================================================ -->
  <filter>
    <filter-name>ResourceRequestFilter</filter-name>
    <filter-class>org.exoplatform.portal.application.ResourceRequestFilter</filter-class>
  </filter>
```

```
  <filter-mapping>
    <filter-name>ResourceRequestFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>


  <!-- ====================================================================== -->
  <!--          LISTENER                                                      -->
  <!-- ====================================================================== -->
  <listener>
    <listener-class>org.exoplatform.container.web.PortalContainerConfigOwner</listener-class>
  </listener>
  <!-- ====================================================================== -->
  <!--          SERVLET                                                       -->
  <!-- ====================================================================== -->
  <servlet>
    <servlet-name>GateInServlet</servlet-name>
    <servlet-class>org.gatein.wci.api.GateInServlet</servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>
  <!--  ====================================================================  -->
  <servlet-mapping>
    <servlet-name>GateInServlet</servlet-name>
    <url-pattern>/gateinservlet</url-pattern>
  </servlet-mapping>
</web-app>
```

How to create/define a portal extension?

A portal extension is in fact a web application declared as a *PortalContainerConfigOwner* (see previous section for more details about a *PortalContainerConfigOwner*) that has been added to the dependency list of the *PortalContainerDefinition* of a given portal.

See below an example of configuration file that add the portal extension "portal-ext" to the dependency list of the portal "portal":

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd          http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
  <external-component-plugins>
```

```
  <!-- The full qualified name of the PortalContainerConfig -->
          <target-component>org.exoplatform.container.definition.PortalContainerConfig</target-
component>
  <component-plugin>
   <!-- The name of the plugin -->
   <name>Add PortalContainer Definitions</name>
    <!-- The name of the method to call on the PortalContainerConfig in order to register the
 PortalContainerDefinitions -->
   <set-method>registerPlugin</set-method>
   <!-- The full qualified name of the PortalContainerDefinitionPlugin -->
   <type>org.exoplatform.container.definition.PortalContainerDefinitionPlugin</type>
   <init-params>
    <object-param>
     <name>portal</name>
     <object type="org.exoplatform.container.definition.PortalContainerDefinition">
      <!-- The name of the portal container -->
      <field name="name"><string>portal</string></field>
      <!-- The name of the context name of the rest web application -->
      <field name="restContextName"><string>rest</string></field>
      <!-- The name of the realm -->
      <field name="realmName"><string>exo-domain</string></field>
      <!-- All the dependencies of the portal container ordered by loading priority -->
      <field name="dependencies">
       <collection type="java.util.ArrayList">
        <value>
         <string>eXoResources</string>
        </value>
        <value>
         <string>portal</string>
        </value>
        <value>
         <string>dashboard</string>
        </value>
        <value>
         <string>exoadmin</string>
        </value>
        <value>
         <string>eXoGadgets</string>
        </value>
        <value>
         <string>eXoGadgetServer</string>
        </value>
        <value>
         <string>rest</string>
```

```
            </value>
            <value>
              <string>web</string>
            </value>
            <value>
              <string>wsrp-producer</string>
            </value>
              <!-- The sample-ext has been added at the end of the dependency list in order to
 have the highest priority towards
            the other web applications and particularly towards "portal" -->
            <value>
              <string>sample-ext</string>
            </value>
          </collection>
        </field>
      </object>
    </object-param>
  </init-params>
 </component-plugin>
</external-component-plugins>
</configuration>
```

How to deploy a portal extension?

Refer to *How to deploy the sample extension?*

How to create/define a new portal?

To duplicate the entire "portal.war" file to create a new portal, you just need to duplicate the following files from the original "portal.war":

- login/jsp/login.jsp
- login/skin: You can customize the css files and pictures
- bookmark.jsp
- favicon.ico: You can replace it by your own logo
- index.jsp
- portal-unavailable.jsp
- portal-warning.jsp
- WEB-INF/web.xml: You just need to change the "display-name" and set a different value for the "realm-name" in the "login-config". Indeed, we must have one realm name per portal.

- WEB-INF/jboss-web.xml: If you use JBoss AS, you need to duplicate also this file and set the new "security-domain" with the new realm name.

You need also to duplicate the "rest.war" file to create a dedicated rest web application for your portal as we must have one rest web application per portal, in fact you just need to duplicate the following files from the original "rest.war":

- WEB-INF/web.xml: You just need to change the "display-name" and set a different value for the "realm-name" in the "login-config". Indeed, we must have one realm name per portal.
- WEB-INF/jboss-web.xml: If you use JBoss AS, you need to duplicate also this file and set the new "security-domain" with the new realm name.

Finally, you need to register and define the corresponding *PortalContainerDefinition*. The *PortalContainerDefinition* of your portal will be composed of:

- The name of new portal
- The name of the context name of the new rest web application
- The name of the new realm
- The list of all the dependencies of the original portal, with the new name of the rest web application instead of the old name (i.e. "rest") and with a new dependency which is in fact the name of your portal. As we leave the dependency of the original portal in the list of dependencies, it will load the configuration files of original "portal.war" file.

See an example below:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd          http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
  <external-component-plugins>
    <!-- The full qualified name of the PortalContainerConfig -->
            <target-component>org.exoplatform.container.definition.PortalContainerConfig</target-
component>
    <component-plugin>
      <!-- The name of the plugin -->
      <name>Add PortalContainer Definitions</name>
        <!-- The name of the method to call on the PortalContainerConfig in order to register the
 PortalContainerDefinitions -->
      <set-method>registerPlugin</set-method>
      <!-- The full qualified name of the PortalContainerDefinitionPlugin -->
      <type>org.exoplatform.container.definition.PortalContainerDefinitionPlugin</type>
      <init-params>
```

```
<object-param>
  <name>sample-portal</name>
  <object type="org.exoplatform.container.definition.PortalContainerDefinition">
    <!-- The name of the portal container -->
    <field name="name"><string>sample-portal</string></field>
    <!-- The name of the context name of the rest web application -->
    <field name="restContextName"><string>rest-sample-portal</string></field>
    <!-- The name of the realm -->
    <field name="realmName"><string>exo-domain-sample-portal</string></field>
    <!-- All the dependencies of the portal container ordered by loading priority -->
    <field name="dependencies">
      <collection type="java.util.ArrayList">
        <value>
          <string>eXoResources</string>
        </value>
        <value>
          <string>portal</string>
        </value>
        <value>
          <string>dashboard</string>
        </value>
        <value>
          <string>exoadmin</string>
        </value>
        <value>
          <string>eXoGadgets</string>
        </value>
        <value>
          <string>eXoGadgetServer</string>
        </value>
        <value>
          <string>rest-sample-portal</string>
        </value>
        <value>
          <string>web</string>
        </value>
        <value>
          <string>wsrp-producer</string>
        </value>
        <value>
          <string>sample-portal</string>
        </value>
      </collection>
    </field>
```

```
        </object>
      </object-param>
    </init-params>
  </component-plugin>
 </external-component-plugins>
</configuration>
```

A portal is implicitly a *PortalContainerConfigOwner* which means that it shares the configuration file embedded into the war file, located at *WEB-INF/conf/configuration.xml*

The position of the servlet context name of this web application in the dependency list is important since the last configuration file loaded has always right towards other configuration files. Each configuration file loaded, could potentially redefine a configuration file that has already been loaded. Moreover, as we now use a unified Servlet Context to load the configuration files, if you want for instance to import the file *war:/conf/database/database-configuration.xml* and this file exists in 2 different web applications, the file from the last (according to the dependency order) web application will be loaded.

How to deploy a new portal?

Refer to *How to deploy the sample portal?*

How to import properly a configuration file using the prefix "war:"?

Now, the *ConfigurationManager* uses by default the unified servlet context of the portal in order to get any resources in particular the configuration files. The unified servlet context is aware of the priorities that has been set in the *PortalContainerDefinition* of the portal. In other words, if you want for instance to import the file *war:/conf/database/database-configuration.xml* and this file exists in 2 different web applications, the file from the last (according to the dependency order) web application will be loaded.

So, in order to avoid issues when we would like to package several products at the same time (i.e. WCM, DMS, CS, KS), we need to:

- Avoid the best you can to redefine a configuration file from the "portal.war" by using the exact same path (like the previous example)
- Add your configuration files in a dedicated folder which name will be the name of the product, in oder to ensure that no other products will use the same path

The example below, is an the example of a file *WEB-INF/conf/configuration.xml* of the product "sample-ext".

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd          http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
  <import>war:/conf/sample-ext/common/common-configuration.xml</import>
  <import>war:/conf/sample-ext/jcr/jcr-configuration.xml</import>
  <import>war:/conf/sample-ext/portal/portal-configuration.xml</import>
  <import>war:/conf/sample-ext/web/web-inf-extension-configuration.xml</import>
</configuration>
```

How to avoid duplicating configuration files just to rename a simple value?

In your configuration file, you can use a special variable called *container.name.suffix* in order to add a suffix to values that could change between portal containers. The value of this variable will be an empty sting if no *PortalContainerDefinition* has been defined otherwise the value will be \-*$portal.container.name*. See an example below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd       http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
   xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
  <component>
    <key>org.exoplatform.services.database.HibernateService</key>
    <jmx-name>database:type=HibernateService</jmx-name>
    <type>org.exoplatform.services.database.impl.HibernateServiceImpl</type>
    <init-params>
      <properties-param>
        <name>hibernate.properties</name>
        <description>Default Hibernate Service</description>
        <property name="hibernate.show_sql" value="false"/>
        <property name="hibernate.cglib.use_reflection_optimizer" value="true"/>
                  <property   name="hibernate.connection.url"   value="jdbc:hsqldb:file:../temp/data/
exodb${container.name.suffix}"/>
        <property name="hibernate.connection.driver_class" value="org.hsqldb.jdbcDriver"/>
        <property name="hibernate.connection.autocommit" value="true"/>
        <property name="hibernate.connection.username" value="sa"/>
        <property name="hibernate.connection.password" value=""/>
```

```
      <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect"/>
      <property name="hibernate.c3p0.min_size" value="5"/>
      <property name="hibernate.c3p0.max_size" value="20"/>
      <property name="hibernate.c3p0.timeout" value="1800"/>
      <property name="hibernate.c3p0.max_statements" value="50"/>
    </properties-param>
  </init-params>
 </component>
</configuration>
```

How to add or change a Repository and/or a Workspace?

Now you can add new JCR repositories or workspaces thanks to an external plugin, the configuration of your JCR Repositories will be merged knowing that the merge algorithm will:

- Add missing Repository Definitions and/or Workspace Definitions.
- Change the properties of a Repository Definition if it has already been defined.
- Replace the Workspace Definition if it has already been defined.

See an example of jcr-configuration.xml below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd          http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
 xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
 <external-component-plugins>
   <!-- The full qualified name of the RepositoryServiceConfiguration -->
         <target-component>org.exoplatform.services.jcr.config.RepositoryServiceConfiguration</
target-component>
   <component-plugin>
     <!-- The name of the plugin -->
     <name>Sample RepositoryServiceConfiguration Plugin</name>
      <!-- The name of the method to call on the RepositoryServiceConfiguration in order to add
 the RepositoryServiceConfigurations -->
     <set-method>addConfig</set-method>
     <!-- The full qualified name of the RepositoryServiceConfigurationPlugin -->
     <type>org.exoplatform.services.jcr.impl.config.RepositoryServiceConfigurationPlugin</type>
     <init-params>
       <value-param>
         <name>conf-path</name>
         <description>JCR configuration file</description>
```

```
        <value>war:/conf/sample-ext/jcr/repository-configuration.xml</value>
      </value-param>
    </init-params>
  </component-plugin>
 </external-component-plugins>
</configuration>
```

See an example of repository-configuration.xml below:

```
<repository-service default-repository="repository">
 <repositories>
      <repository name="repository" system-workspace="system" default-workspace="portal-system">
    <security-domain>exo-domain</security-domain>
    <access-control>optional</access-control>
      <authentication-policy>org.exoplatform.services.jcr.impl.core.access.JAASAuthenticator</authentication-policy>
    <workspaces>
     <workspace name="sample-ws">
                                                                                <container class="org.exoplatform.services.jcr.impl.storage.jdbc.JDBCWorkspaceDataContainer">
        <properties>
         <property name="source-name" value="jdbcexo${container.name.suffix}" />
         <property name="dialect" value="hsqldb" />
         <property name="multi-db" value="false" />
         <property name="update-storage" value="true" />
         <property name="max-buffer-size" value="204800" />
                                <property name="swap-directory" value="../temp/swap/sample-ws${container.name.suffix}" />
        </properties>
        <value-storages>
                                                                    <value-storage id="sample-ws" class="org.exoplatform.services.jcr.impl.storage.value.fs.TreeFileValueStorage">
          <properties>
           <property name="path" value="../temp/values/sample-ws${container.name.suffix}" />
          </properties>
          <filters>
           <filter property-type="Binary" />
          </filters>
         </value-storage>
        </value-storages>
       </container>
       <initializer class="org.exoplatform.services.jcr.impl.core.ScratchWorkspaceInitializer">
```

```
          <properties>
            <property name="root-nodetype" value="nt:unstructured" />
            <property name="root-permissions"
              value="any read;*:/platform/administrators read;*:/platform/administrators add_node;*:/
platform/administrators set_property;*:/platform/administrators remove" />
          </properties>
        </initializer>
        <cache enabled="true">
          <properties>
            <property name="max-size" value="20000" />
            <property name="live-time" value="30000" />
          </properties>
        </cache>
        <query-handler class="org.exoplatform.services.jcr.impl.core.query.lucene.SearchIndex">
          <properties>
                                  <property name="index-dir" value="../temp/jcrlucenedb/sample-
ws${container.name.suffix}" />
          </properties>
        </query-handler>
        <lock-manager>
          <time-out>15m</time-out><!-- 15min -->
          <persister class="org.exoplatform.services.jcr.impl.core.lock.FileSystemLockPersister">
            <properties>
              <property name="path" value="../temp/lock/sample-ws${container.name.suffix}" />
            </properties>
          </persister>
        </lock-manager>
      </workspace>
    </workspaces>
  </repository>
 </repositories>
</repository-service>
```

If you have to change the default repository or the default workspace, please be careful since it could cause side effects as you could be incompatible with some portal configuration files that refer explicitly to *portal-system* and/or to *repository*. To solve this problem you can either redefine the configuration file in a portal extension to change the workspace name and/or the repository name or you could also add your own repository instead of your own workspace.

How to add new ResourceBundles to my portal?

Now you can add new Resource Bundles, thanks to an external plugin.

See an example below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd          http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
  <external-component-plugins>
    <!-- The full qualified name of the ResourceBundleService -->
          <target-component>org.exoplatform.services.resources.ResourceBundleService</target-
component>
    <component-plugin>
      <!-- The name of the plugin -->
      <name>Sample ResourceBundle Plugin</name>
       <!-- The name of the method to call on the ResourceBundleService in order to register the
 ResourceBundles -->
      <set-method>addResourceBundle</set-method>
      <!-- The full qualified name of the BaseResourceBundlePlugin -->
      <type>org.exoplatform.services.resources.impl.BaseResourceBundlePlugin</type>
      <init-params>
        <!--values-param>
          <name>classpath.resources</name>
            <description>The resources that start with the following package name should be load
 from file system</description>
          <value>locale.portlet</value>
        </values-param-->
        <values-param>
          <name>init.resources</name>
          <description>Store the following resources into the db for the first launch </description>
          <value>locale.portal.sample</value>
        </values-param>
        <values-param>
          <name>portal.resource.names</name>
          <description>The properties files of the portal , those file will be merged
            into one ResoruceBundle properties </description>
          <value>locale.portal.sample</value>
        </values-param>
      </init-params>
    </component-plugin>
```

```
  </external-component-plugins>
</configuration>
```

How to overwrite existing ResourceBundles in my portal?

Now each portal container has its own *ClassLoader* which is automatically set for you at runtime (FYI: it could be retrieved thanks to *portalContainer.getPortalClassLoader()*). This *ClassLoader* is an unified *ClassLoader* that is also aware of the dependency order defined into the *PortalContainerDefinition*, so to add new keys or update key values, you just need to:

- Add the corresponding resource bundle with the same name, with the same extension (xml or properties) at the same location into the *classpath* of your Web application (i.e. directly into the *WEB-INF/classes* directory or into a jar file in the *WEB-INF/lib* directory)
- Ensure that your web application is defined after the web application of the portal in the dependency list of the related *PortalContainerDefinition*.

In the example below, we want to change the values of the keys *UIHomePagePortlet.Label.Username* and *UIHomePagePortlet.Label.Password*, and add the new key *UIHomePagePortlet.Label.SampleKey* into the Resource Bundle *locale.portal.webui*.

WEB-INF/classes/local/portal/webui_en.properties

```
###############################################################################
#org.exoplatform.portal.webui.component.UIHomePagePortlet                    #
###############################################################################

UIHomePagePortlet.Label.Username=Usr:
UIHomePagePortlet.Label.Password=Pwd:
UIHomePagePortlet.Label.SampleKey=This is a new key that has been added to the
 Resource Bundle "locale.portal.webui" of "sample-ext"
```

How to replace a groovy template of my portal?

Now each portal container has its own *ServletContext* which is automatically set for you at runtime (FYI: it could be retrieved thanks to *portalContainer.getPortalContext()*). This *ServletContext* is an unified *ServletContext* that is also aware of the dependency order defined into the *PortalContainerDefinition* so to replace a groovy template of the portal, you just need to:

- Add the corresponding groovy template with the same name at the same location into your Web application
- Ensure that your web application is defined after the web application of the portal in the dependency list of the related *PortalContainerDefinition*.

How to add new Portal Configurations, Navigations, Pages or Portlet Preferences to my portal?

Now you can add new Portal Configurations, Navigations, Pages or Portlet Preferences thanks to an external plugin.

See an example below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd          http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
  <external-component-plugins>
    <!-- The full qualified name of the UserPortalConfigService -->
                  <target-component>org.exoplatform.portal.config.UserPortalConfigService</target-
component>
    <component-plugin>
      <!-- The name of the plugin -->
      <name>new.portal.config.user.listener</name>
       <!-- The name of the method to call on the UserPortalConfigService in order to register the
 NewPortalConfigs -->
      <set-method>initListener</set-method>
      <!-- The full qualified name of the NewPortalConfigListener -->
      <type>org.exoplatform.portal.config.NewPortalConfigListener</type>
      <description>this listener init the portal configuration</description>
      <init-params>
        <object-param>
          <name>portal.configuration</name>
          <description>description</description>
          <object type="org.exoplatform.portal.config.NewPortalConfig">
            <field name="predefinedOwner">
              <collection type="java.util.HashSet">
                <value>
                  <string>classic</string>
                </value>
              </collection>
            </field>
            <field name="ownerType">
              <string>portal</string>
            </field>
            <field name="templateLocation">
              <string>war:/conf/sample-ext/portal</string>
```

```
          </field>
        </object>
      </object-param>
      <object-param>
        <name>group.configuration</name>
        <description>description</description>
        <object type="org.exoplatform.portal.config.NewPortalConfig">
          <field name="predefinedOwner">
            <collection type="java.util.HashSet">
              <value>
                <string>platform/users</string>
              </value>
            </collection>
          </field>
          <field name="ownerType">
            <string>group</string>
          </field>
          <field name="templateLocation">
            <string>war:/conf/sample-ext/portal</string>
          </field>
        </object>
      </object-param>
      <object-param>
        <name>user.configuration</name>
        <description>description</description>
        <object type="org.exoplatform.portal.config.NewPortalConfig">
          <field name="predefinedOwner">
            <collection type="java.util.HashSet">
              <value>
                <string>root</string>
              </value>
            </collection>
          </field>
          <field name="ownerType">
            <string>user</string>
          </field>
          <field name="templateLocation">
            <string>war:/conf/sample-ext/portal</string>
          </field>
        </object>
      </object-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

```
</configuration>
```

How to add new Http Filters to my portal without modifying the portal binary?

We added a *GenericFilter* that allows you to define new Http Filters thanks to an external plugin. Your filter will need to implement the interface *org.exoplatform.web.filter.Filter*.

See an example of configuration below:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd          http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
  <external-component-plugins>
    <!-- The full qualified name of the ExtensibleFilter -->
    <target-component>org.exoplatform.web.filter.ExtensibleFilter</target-component>
    <component-plugin>
      <!-- The name of the plugin -->
      <name>Sample Filter Definition Plugin</name>
          <!-- The name of the method to call on the ExtensibleFilter in order to register the
 FilterDefinitions -->
      <set-method>addFilterDefinitions</set-method>
      <!-- The full qualified name of the FilterDefinitionPlugin -->
      <type>org.exoplatform.web.filter.FilterDefinitionPlugin</type>
      <init-params>
        <object-param>
          <name>Sample Filter Definition</name>
          <object type="org.exoplatform.web.filter.FilterDefinition">
            <!-- The filter instance -->
          <field name="filter"><object type="org.exoplatform.sample.ext.web.SampleFilter"/></field>
            <!-- The mapping to use -->
            <!-- WARNING: the mapping is expressed with regular expressions -->
            <field name="patterns">
              <collection type="java.util.ArrayList" item-type="java.lang.String">
                <value>
                  <string>/.*</string>
                </value>
              </collection>
            </field>
          </object>
```

```
      </object-param>
    </init-params>
  </component-plugin>
 </external-component-plugins>
</configuration>
```

See an example of Filter below:

SampleFilter.java

```
...
import org.exoplatform.web.filter.Filter;

import java.io.IOException;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class SampleFilter implements Filter
{

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain
 chain) throws IOException,
      ServletException
  {
     System.out.println("SampleFilter start");
     try
     {
        chain.doFilter(request, response);
     }
     finally
     {
        System.out.println("SampleFilter end");
     }
  }
}
```

How to add new *HttpSessionListeners* and/or *ServletContextListeners* to my portal without modifying the portal binary?

We added a *GenericHttpListener* that allows you to define new *HttpSessionListeners* and/or *ServletContextListeners* thanks to an external plugin. Actually, the *GenericHttpListener* will broadcast events thanks to the *ListenerService* that you can easily capture. The events that it broadcasts are:

- *org.exoplatform.web.GenericHttpListener.sessionCreated*: When a new session is created in the portal.
- *org.exoplatform.web.GenericHttpListener.sessionDestroyed*: When a session is destroyed in the portal.
- *org.exoplatform.web.GenericHttpListener.contextInitialized*: When the servlet context of the portal is initialized.
- *org.exoplatform.web.GenericHttpListener.contextDestroyed*: When the servlet context of the portal is destroyed.

If you want to listen to *org.exoplatform.web.GenericHttpListener.sessionCreated*, you will need to create a Listener that extends _Listener<PortalContainer, HttpSessionEvent>_If you want to listen to _org.exoplatform.web.GenericHttpListener.sessionDestroyed_, you will need to create a Listener that extends _Listener<PortalContainer, HttpSessionEvent>_If you want to listen to _org.exoplatform.web.GenericHttpListener.contextInitialized_, you will need to create a Listener that extends _Listener<PortalContainer, ServletContextEvent>_If you want to listen to _org.exoplatform.web.GenericHttpListener.contextDestroyed_, you will need to create a Listener that extends *Listener<PortalContainer, ServletContextEvent>*

See an example of configuration below:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd          http://
www.exoplatform.org/xml/ns/kernel_1_2.xsd"
  xmlns="http://www.exoplatform.org/xml/ns/kernel_1_2.xsd">
  <external-component-plugins>
    <!-- The full qualified name of the ListenerService -->
    <target-component>org.exoplatform.services.listener.ListenerService</target-component>
    <component-plugin>
      <!-- The name of the listener that is also the name of the target event -->
      <name>org.exoplatform.web.GenericHttpListener.sessionCreated</name>
```

```
    <!-- The name of the method to call on the ListenerService in order to register the Listener -->
    <set-method>addListener</set-method>
    <!-- The full qualified name of the Listener -->
    <type>org.exoplatform.sample.ext.web.SampleHttpSessionCreatedListener</type>
  </component-plugin>
  <component-plugin>
    <!-- The name of the listener that is also the name of the target event -->
    <name>org.exoplatform.web.GenericHttpListener.sessionDestroyed</name>
   <!-- The name of the method to call on the ListenerService in order to register the Listener -->
    <set-method>addListener</set-method>
    <!-- The full qualified name of the Listener -->
    <type>org.exoplatform.sample.ext.web.SampleHttpSessionDestroyedListener</type>
  </component-plugin>
  <component-plugin>
    <!-- The name of the listener that is also the name of the target event -->
    <name>org.exoplatform.web.GenericHttpListener.contextInitialized</name>
   <!-- The name of the method to call on the ListenerService in order to register the Listener -->
    <set-method>addListener</set-method>
    <!-- The full qualified name of the Listener -->
    <type>org.exoplatform.sample.ext.web.SampleContextInitializedListener</type>
  </component-plugin>
  <component-plugin>
    <!-- The name of the listener that is also the name of the target event -->
    <name>org.exoplatform.web.GenericHttpListener.contextDestroyed</name>
   <!-- The name of the method to call on the ListenerService in order to register the Listener -->
    <set-method>addListener</set-method>
    <!-- The full qualified name of the Listener -->
    <type>org.exoplatform.sample.ext.web.SampleContextDestroyedListener</type>
  </component-plugin>
 </external-component-plugins>
</configuration>
```

See an example of Session Listener below:

SampleHttpSessionCreatedListener.java

```
..
import org.exoplatform.container.PortalContainer;
import org.exoplatform.services.listener.Event;
import org.exoplatform.services.listener.Listener;

import javax.servlet.http.HttpSessionEvent;
```

```
public class SampleHttpSessionCreatedListener extends Listener<PortalContainer,
 HttpSessionEvent>
{

  @Override
     public void onEvent(Event<PortalContainer, HttpSessionEvent> event) throws
 Exception
  {
    System.out.println("Creating a new session");
  }
}
```

See an example of Context Listener below:

SampleContextInitializedListener.java

```
..
import org.exoplatform.container.PortalContainer;
import org.exoplatform.services.listener.Event;
import org.exoplatform.services.listener.Listener;

import javax.servlet.ServletContextEvent;

public class SampleContextInitializedListener extends Listener<PortalContainer,
 ServletContextEvent>
{

  @Override
     public void onEvent(Event<PortalContainer, ServletContextEvent> event) throws
 Exception
  {
    System.out.println("Initializing the context");
  }
}
```

How to add new *HttpServlet* to my portal without modifying the portal binary?

Actually, it is not possible but you can create a rest component instead. For more details about rest, please refer to the following article *WS*.

How to override or add a Context Parameter to my portal without modifying the portal binary?

Actually, you have nothing to do, you just need to ensure that you get the context parameter value from the unified servlet context of the portal, that should be set for you but you can still get it from *portalContainer.getPortalContext()*.

Where can I found an example of how to extend my portal?

We added an example of portal extension (i.e. ability to customize a portal without changing anything in the portal.ear) that you can find in the svn of gatein at gatein/sample/extension.

How to deploy the sample extension?

On JBoss (tested on JBoss 5.1.0.GA)

We assume that you have a clean JBoss version of GateIn, in other words, we assume that you have already the file *exoplatform.ear* in the *deploy* directory of JBoss and you have the related application policy in your *conf/login-config.xml*.

You need to:

- Add the file *sample-ext.ear* from sample/extension/ear/target/ to the deploy directory of JBoss, this file contains:
  - The file *sample-ext.war* which is the web application that contains (potentially) configuration files, groovy templates, resource bundles, skins and javascript files, that will extend the portal.
  - The file *exo.portal.sample.extension.config-X.Y.Z.jar* which is the file in which we defined the *PortalContainerDefinition* of the original portal in which we added *sample-ext* at the end of dependency list.
  - The file *exo.portal.sample.extension.jar-X.Y.Z.jar* which is the file in which we have internal classes that are actualy a set of sample classes *(SampleFilter, SampleContextInitializedListener, SampleContextDestroyedListener, SampleHttpSessionCreatedListener and SampleHttpSessionDestroyedListener)*
- Add the file *starter.ear* from starter/ear/target/ to the deploy directory of JBoss, this file contains:
  - The file *starter.war* which is the web application that will create and start all the portal containers, that is why it must be launched after all the other web applications.

This can only work if a Unified ClassLoader has been configured on your JBoss (default behavior) and the load order is first the *exoplatform.ear* then the *sample-ext.ear* and finally the *starter.ear*.

The file *starter.ear* must always been started last.

On Tomcat (tested on Tomcat 6.0.20)

We assume that you have a clean Tomcat version of GateIn, in other words, we assume that you have already all the jar files of GateIn and their dependencies into *tomcat/lib*, you have all the war files of GateIn into *tomcat/webapps* and you have the realm name "exo-domain" defined into the file *tomcat/conf/jaas.conf*.

- Add the file *sample-ext.war* from sample/extension/war/target/ to the *tomcat/webapps* directory. *(See the purpose of this file in the JBoss section)*.
- Add the folder *starter* from starter/war/target/ to the *tomcat/webapps* directory. *(See the purpose of this file in the JBoss section)*.
- Rename the directory (unzipped folder) *starter* to *starter.war* (for more details see the warning below)
- Add the jar file *exo.portal.sample.extension.config-X.Y.Z.jar* from sample/extension/config/target/ to the *tomcat/lib* directory. *(See the purpose of this file in the JBoss section)*.
- Add the jar file *exo.portal.sample.extension.jar-X.Y.Z.jar* from sample/extension/jar/target/ to the *tomcat/lib* directory. *(See the purpose of this file in the JBoss section)*.

This can only work if the *starter.war* is the last war file to be loaded, so don't hesitate to rename it if your war files are loaded following to the alphabetic order.

Where can I find an example of how to create a new portal?

We added an example of new portal (i.e. ability to create a new portal from another portal without changing anything in the portal.ear) that you can find in the svn of gatein at gatein/sample/portal.

How to deploy the sample portal?

On JBoss (tested on JBoss 5.1.0.GA)

We assume that you have a clean JBoss version of GateIn, in other words, we assume that you have already the file *exoplatform.ear* in the *deploy* directory of JBoss and you have the related application policy in your *conf/login-config.xml*.

You need to:

- Add the file *sample-portal.ear* from *sample/portal/ear/target/* to the deploy directory of JBoss, this file contains:

- The file *sample-portal.war* which is the entry point of the new portal
- The file *rest-sample-portal.war* which is the entry point for *rest* outside the portal (in the portal you can access to *rest* thanks to path prefix */sample-portal/rest*)
- The file *exo.portal.sample.portal.config-X.Y.Z.jar* which is the file in which we defined the *PortalContainerDefinition* of this portal
- The file *exo.portal.sample.portal.jar-X.Y.Z.jar* which is the file in which we have internal classes that are actualy a set of sample classes *(SampleFilter, SampleContextInitializedListener, SampleContextDestroyedListener, SampleHttpSessionCreatedListener and SampleHttpSessionDestroyedListener)*

- Add the file *starter.ear* from starter/ear/target/ to the deploy directory of JBoss, this file contains:
  - The file *starter.war* which is the web application that will create and start all the portal containers, that is why it must be launched after all the other web applications.

- Define the related application policy in your file *conf/login-config.xml*, as below:

```
 <application-policy name="exo-domain-sample-portal">
   <authentication>
     <login-module code="org.exoplatform.web.security.PortalLoginModule" flag="required">
       <module-option name="portalContainerName">sample-portal</module-option>
       <module-option name="realmName">exo-domain-sample-portal</module-option>
     </login-module>
           <login-module  code="org.exoplatform.services.security.jaas.SharedStateLoginModule"
flag="required">
       <module-option name="portalContainerName">sample-portal</module-option>
       <module-option name="realmName">exo-domain-sample-portal</module-option>
     </login-module>
                  <login-module    code="org.exoplatform.services.security.j2ee.JbossLoginModule"
flag="required">
       <module-option name="portalContainerName">sample-portal</module-option>
       <module-option name="realmName">exo-domain-sample-portal</module-option>
     </login-module>
   </authentication>
 </application-policy>
```

This can only work if a Unified ClassLoader has been configured on your JBoss (default behavior) and the load order is first the *exoplatform.ear* then the *sample-portal.ear* and finally the *starter.ear*.

The file *starter.ear* must always been started last.

On Tomcat (tested on Tomcat 6.0.20)

We assume that you have a clean Tomcat version of GateIn, in other words, we assume that you have already all the jar files of GateIn and their dependencies into *tomcat/lib*, you have all the war files of GateIn into *tomcat/webapps* and you have the realm name "exo-domain" defined into the file *tomcat/conf/jaas.conf*.

- Add the file *sample-portal.war* from sample/portal/war/target/ to the *tomcat/webapps* directory. *(See the purpose of this file in the JBoss section)*.
- Add the file *rest-sample-portal.war* from sample/portal/rest-war/target/ to the *tomcat/webapps* directory. *(See the purpose of this file in the JBoss section)*.
- Add the folder *starter* from starter/war/target/ to the *tomcat/webapps* directory. *(See the purpose of this file in the JBoss section)*.
- Rename the directory (unzipped folder) *starter* to *starter.war (for more details see the warning below)*.
- Add the jar file *exo.portal.sample.portal.config-X.Y.Z.jar* from sample/portal/config/target/ to the *tomcat/lib* directory. *(See the purpose of this file in the JBoss section)*.
- Add the jar file *exo.portal.sample.portal.jar-X.Y.Z.jar* from sample/portal/jar/target/ to the *tomcat/lib* directory. *(See the purpose of this file in the JBoss section)*.
- Define the related realm in your file *tomcat/conf/jaas.conf*, as below:

tomcat/conf/jaas.conf

```
...
exo-domain-sample-portal {
  org.exoplatform.web.security.PortalLoginModule required
    portalContainerName="sample-portal"
    realmName="exo-domain-sample-portal";
  org.exoplatform.services.security.jaas.SharedStateLoginModule required
    portalContainerName="sample-portal"
    realmName="exo-domain-sample-portal";
  org.exoplatform.services.security.j2ee.TomcatLoginModule required
    portalContainerName="sample-portal"
    realmName="exo-domain-sample-portal";
};
```

- Define the context of *sample-portal* by creating a file called *sample-portal.xml* into *tomcat/conf/Catalina/localhost/* with the following content:

tomcat/conf/Catalina/localhost/sample-portal.xml

```
<Context path='/sample-portal' docBase='sample-portal' debug='0' reloadable='true'
 crossContext='true' privileged='true'>
  <Logger className='org.apache.catalina.logger.SystemOutLogger'
          prefix='localhost_portal_log.' suffix='.txt' timestamp='true'/>
                <Manager          className='org.apache.catalina.session.PersistentManager'
 saveOnRestart='false'/>
  <Realm className='org.apache.catalina.realm.JAASRealm'
         appName='exo-domain-sample-portal'
         userClassNames='org.exoplatform.services.security.jaas.UserPrincipal'
         roleClassNames='org.exoplatform.services.security.jaas.RolePrincipal'
         debug='0' cache='false'/>
              <Valve          className='org.apache.catalina.authenticator.FormAuthenticator'
 characterEncoding='UTF-8'/></Context>
```

- Define the context of *rest-sample-portal* by creating a file called *rest-sample-portal.xml* into *tomcat/conf/Catalina/localhost/* with the following content:

tomcat/conf/Catalina/localhost/rest-sample-portal.xml

```
<Context path="/rest-sample-portal" docBase="rest-sample-portal" reloadable="true"
 crossContext="false">

  <Logger className='org.apache.catalina.logger.SystemOutLogger'
          prefix='localhost_portal_log.' suffix='.txt' timestamp='true'/>
                <Manager      className='org.apache.catalina.session.PersistentManager'
 saveOnRestart='false'/>
    <Realm className='org.apache.catalina.realm.JAASRealm'
         appName='exo-domain-sample-portal'
         userClassNames="org.exoplatform.services.security.jaas.UserPrincipal"
         roleClassNames="org.exoplatform.services.security.jaas.RolePrincipal"
         debug='0' cache='false'/>
</Context>
```

This can only work if the *starter.war* is the last war file to be loaded, so don't hesitate to rename it if your war files are loaded following to the alphabetic order.

I get "java.lang.IllegalStateException: No pre init tasks can be added to the portal container 'portal', because it has already been initialized." what can I do to fix it?

To fix this issue you need to check if:

1. The file *starter-gatein.ear* (which will be *starter.war* for Tomcat) has been deployed
2. The file *starter-gatein.ear* (which will be *starter.war* for Tomcat) is the last ear file to be launched

Note

With Tomcat to prevent any alphabetic issue, the good way to solve this problem is to:

- Unzip the archive *starter.war* into a directory called *starter*
- Remove the archive *starter.war*
- Rename the folder *starter* to *starter.war*

This tip works because folders corresponding to unzipped wars are launched after war files.

Recommendations

Don't ship your configuration files with your jar files?

Remove all the configuration files from the jar files (*conf/configuration.xml* and *conf/portal/configuration.xml*) and move them to the war file of your extension, otherwise your configuration files will be loaded for all the portal containers which could cause incompatibility issues with other portals.

Each extension should manage independently, its css files, js files, google gadgets and configuration files. If you add configuration files into the jar files of your extension, you brake this law.

Using a dedicated workspace/repository for your extension?

In order to avoid conflicts with other extensions and to manage each extension independently, it is highly recommended to use a dedicated workspace or repository per extension.

How to use AS Managed DataSource under JBoss AS

Configurations Steps

Checked under Gatein 3.1.0-GA Final

only no-tx-datasource is supported in JCR 1.12

Declaring the datasources in the AS

Under JBoss, just put a file XXX-ds.xml in the deploy server (example: \server\default\deploy). In this file, we will configure all datasources which eXo will need. (there should be 4 named: jdbcjcr_portal, jdbcjcr_portal-sample, jdbcidm_portal & jdbcidm_sample-portal).

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <no-tx-datasource>
    <jndi-name>jdbcjcr_portal</jndi-name>
    <connection-url>jdbc:hsqldb:${jboss.server.data.dir}/data/jdbcjcr_portal</connection-url>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <user-name>sa</user-name>
    <password></password>
  </no-tx-datasource>

  <no-tx-datasource>
    <jndi-name>jdbcjcr_sample-portal</jndi-name>
   <connection-url>jdbc:hsqldb:${jboss.server.data.dir}/data/jdbcjcr_sample-portal</connection-
url>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <user-name>sa</user-name>
    <password></password>
  </no-tx-datasource>

  <no-tx-datasource>
    <jndi-name>jdbcidm_portal</jndi-name>
```

```
    <connection-url>jdbc:hsqldb:${jboss.server.data.dir}/data/jdbcidm_portal</connection-url>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <user-name>sa</user-name>
    <password></password>
  </no-tx-datasource>

  <no-tx-datasource>
    <jndi-name>jdbcidm_sample-portal</jndi-name>
                <connection-url>jdbc:hsqldb:${jboss.server.data.dir}/data/jdbcidm_sample-portal</
connection-url>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <user-name>sa</user-name>
    <password></password>
  </no-tx-datasource>
</datasources>
```

Which properties can be set for datasource can be found here: *Configuring JDBC DataSources - The non transactional DataSource configuration schema* [http://docs.jboss.org/jbossas/docs/Server_Configuration_Guide/4/html/Connectors_on_JBoss-Configuring_JDBC_DataSources.html#Configuring_JDBC_DataSources-The_non_transactional_DataSource_configuration_schema]

Do not let eXo bind datasources explicitly

Edit server/default/conf/gatein/configuration.properties and comment out next rows in JCR section:

```
#gatein.jcr.datasource.driver=org.hsqldb.jdbcDriver
#gatein.jcr.datasource.url=jdbc:hsqldb:file:${gatein.db.data.dir}/data/jdbcjcr_${name}
#gatein.jcr.datasource.username=sa
#gatein.jcr.datasource.password=
```

and in IDM section:

```
#gatein.idm.datasource.driver=org.hsqldb.jdbcDriver
#gatein.idm.datasource.url=jdbc:hsqldb:file:${gatein.db.data.dir}/data/jdbcidm_${name}
#gatein.idm.datasource.username=sa
#gatein.idm.datasource.password=
```

In jcr-configuration.xml and idm-configuration.xml comment out the plugin InitialContextInitializer.

```
<!-- Commented because, Datasources are declared and bound by AS, not in eXo -->
<!--
<external-component-plugins>
    [...]
</external-component-plugins>
-->
```

Running eXo after these configurations goes well.

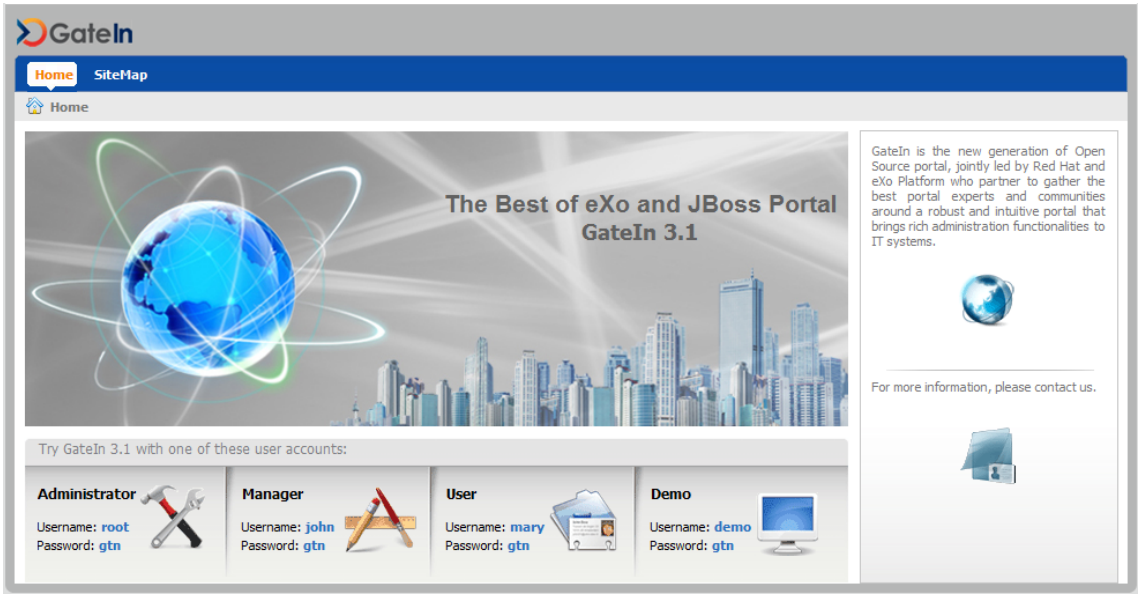
GateIn Reference Guide

the GateIn community , JBoss by Red Hat , and eXo Platform

Scott Mumford, Thomas Heute, Luc Texier, and Christophe Laprun

Introduction

GateIn 3.2 is the merge of two mature Java projects; JBoss Portal and eXo Portal. This new community project takes the best of both offerings and incorporates them into a single portal framework. GateIn aims at providing an intuitive user-friendly portal, and a framework to address the needs of Web 2.0 applications today.



This book provides thorough information about installation and configuration of the services provided by GateIn.

Related Links

- GateIn homepage: *www.gatein.org* [http://www.gatein.org]
- GateIn videos: *vimeo.com/channels/gatein* [http://vimeo.com/channels/gatein]
- GateIn documentation: *www.jboss.org/gatein/documentation.html* [http://www.jboss.org/gatein/documentation.html]
- GateIn downloads: *www.jboss.org/gatein/downloads.html* [http://www.jboss.org/gatein/downloads.html]

Configuration

Database Configuration

Overview

GateIn 3.2 has two different database dependencies. One is the identity service configuration, which depends on Hibernate. The other is the Java content repository (JCR) service, which depends on JDBC API, and can integrate with any existing datasource implementation.

When you change the database configuration for the first time, GateIn will automatically generate the proper schema (assuming that the database user has the appropriate permissions).

GateIn 3.2 assumes the default encoding for your database is `latin1`. You may need to change this parameter for your database so that GateIn 3.2 works properly.

Configuring the database for JCR

To configure the database used by JCR, edit the file:

```
$JBOSS_HOME/server/default/conf/gatein/configuration.properties
```

.

For Tomcat, the file is located at

```
$TOMCAT_HOME/gatein/conf/configuration.properties
```

.

And edit the values of driver, url, username and password with the values for your JDBC connection. To learn more, refer to your database JDBC driver documentation.

```
gatein.jcr.datasource.driver=org.hsqldb.jdbcDriver
gatein.jcr.datasource.url=jdbc:hsqldb:file:${gatein.db.data.dir}/data/jdbcjcr_${name}
gatein.jcr.datasource.username=sa
gatein.jcr.datasource.password=
```

By default, the name of the database is "jdbcjcr_${name}" - ${name} should be a part of the database name, as it is dynamically replaced by the name of the portal container extension. For example, gatein-sample-portal.ear defines "sample-portal" as the container name and the default portal defines "portal" as the container name).

In case of HSQL, the databases are created automatically. For any other databases, you need to create a database named jdbcjcr_portal (and "jdbcjcr_sample-portal" if you have gatein-sample-portal.ear in $JBOSS_HOME/server/default/deploy). Note that some databases do not accept '-' in the database name, so you may have to remove $JBOSS_HOME/server/default/deploy/gatein-sample-portal.ear.

Make sure the user has rights to create tables on jdbcjcr_portal, and to update them as they will be automatically created during the first startup.

Also, add your database's JDBC driver into the classpath - you can put it in $JBOSS_HOME/server/default/lib (or $TOMCAT_HOME/lib, if you are running on Tomcat).

MySQL example:

Configure the JCR to store data in MySQL and assume that you created a user named "gateinuser" with a password "gateinpassword". Next, create a database "mygateindb_portal" (remember that portal is required), and assign the rights to create tables to the users.

Add the MySQL's JDBC driver to the classpath, and finally edit gatein.ear/02portal.war/WEB-INF/conf/jcr/jcr-configuration to contain the following:

```
gatein.jcr.datasource.driver=com.mysql.jdbc.Driver
gatein.jcr.datasource.url=jdbc:mysql://localhost:3306/mygateindb${container.name.suffix}
gatein.jcr.datasource.username=gateinuser
gatein.jcr.datasource.password=gateinpassword
```

Configuring the database for the default identity store

By default, users are stored in a database. To change the database where users are stored, you need to edit the file:

```
$JBOSS_HOME/server/default/conf/gatein/configuration.properties
```

For Tomcat, the file is located at

```
$TOMCAT_HOME/gatein/conf/configuration.properties
```

You will find the same type of configuration as in jcr-configuration.xml:

```
gatein.idm.datasource.driver=org.hsqldb.jdbcDriver
gatein.idm.datasource.url=jdbc:hsqldb:file:${gatein.db.data.dir}/data/jdbcidm_${name}
gatein.idm.datasource.username=sa
```

```
gatein.idm.datasource.password
```

Email Service Configuration

Overview

GateIn 3.2 includes an e-mail sending service that needs to be configured before it can function properly. This service, for instance, is used to send emails to users who forgot their password or username.

Configuring the outgoing e-mail account

The e-mail service can use any SMTP account configured in $JBOSS_HOME/server/default/conf/gatein/configuration.properties (or $TOMCAT_HOME/gatein/conf/configuration.properties if you are using Tomcat).

The relevant section looks like:

```
# EMail
gatein.email.smtp.username=
gatein.email.smtp.password=
gatein.email.smtp.host=smtp.gmail.com
gatein.email.smtp.port=465
gatein.email.smtp.starttls.enable=true
gatein.email.smtp.auth=true
gatein.email.smtp.socketFactory.port=465
gatein.email.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory
```

It is preconfigured for GMail, so that any GMail account can easily be used (simply use the full GMail address with username and password.

In corporate environments, if you want to use your corporate SMTP gateway over SSL, like in the default configuration, configure a certificate truststore containing your SMTP server's public certificate. Depending on the key sizes, you may then also need to install Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files for your Java Runtime Environment.

Portal Development

Skinning the portal

Overview

GateIn 3.2 provides robust skinning support for the entire portal User Interface (UI). This includes support for skinning all of the common portal elements and being able to provide custom skins and window decoration for individual portlets. It is designed to common graphic resource reuse and ease of development .

Skin Components

The complete skinning of a page can be decomposed into three main parts:

Portal Skin
: The portal skin contains the CSS styles for the portal and its various UI components. This should include all the UI components, except for the window decorators and portlet specific styles.

Window Styles
: The CSS styles are associated with the porlet window decorators. The window decorators contain the control buttons and boarders surrounding each portlet. Individual portlets can have their own window decorator selected, or be rendered without one.

Portlet Skins
: The portlet skins affects how portlets are rendered on the page via one of the following ways:

  Portlet Specification CSS Classes
  : The portlet specification defines a set of CSS classes that should be available to portlets. GateIn 3.2 provides these classes as part of the portal skin. This allows each portal skin to define its own look and feel for these default values.

  Portlet Skins
  : GateIn 3.2 provides a means for portlet CSS files to be loaded, basing on the current portal skin. This allows a portlet to provide different CSS styles to better match the look and feel of the current portal. Portlet skins provide a much more customizable CSS experience than just using the portlet specification CSS classes.

Note

The window decorators and the default portlet specification CSS classes should be considered as separate types of skinning components, but they need to be included as part of the overall portal skin. The portal skin must include these component´s CSS classes or they will not be displayed correctly.

A portlet skin does not need to be included as part of the portal skin and can be included within the portlets web application.

Skin Selection

Skin Selection Through the User Interface

There are a few means for you to select the display skin. The easiest way to change the skin is to select it through the user interface. Admininistrators can change the default skin for the portal, and users logged in can select their desired display skins.

Please see the User Guide for information on how to change the skin using the user interface.

Setting the Default Skin within the Configuration Files

The default skin can also be configured through the portal configuration files if you do not want to use the admin user interface. This will allow the portal to have the new default skin ready when GateIn 3.2 is initially started.

The default skin of the portal is called `Default`. To change this value, add a `skin` tag in the `02portal.war/WEB-INF/conf/portal/portal/classic/portal.xml` configuration file.

To change the skin to `MySkin`, you would make the following changes:

```
<portal-config>
        <portal-name>classic</portal-name>
        <locale>en</locale>
        <access-permissions>Everyone</access-permissions>
        <edit-permission>*:/platform/administrators</edit-permission>
        <skin>MySkin</skin>
        ...
```

Skins in Page Markups

The GateIn 3.2 skin contains CSS styles for the portal's components but also shares components that may be reused in portlets. When GateIn 3.2 generates a portal page markup, it inserts the stylesheet links in the page's `head` tag.

There are two main types of CSS links that will appear in the `head` tag: a link to the portal skin CSS file and a link to the portlet skin CSS files.

Portal Skin
: The portal skin will appear as a single link to a CSS file. This link contains contents from all the portal skin classes merged into one file. This allows the portal skin to be transfered more

quickly as a single file instead of many multiple smaller files. All pages of the portal have the same skin defined in the CSS file.

Portlet Skin
: Each portlet on a page may contribute its own style. The link to the portlet skin will only appear on the page if that portlet is loaded on the current page. A page may contain many portlet skin CSS links or none.

In the code fragment below, you can see two types of links:

```
<head>
...
<!-- The portal skin -->
<link id="CoreSkin" rel="stylesheet" type="text/css" href="/eXoResources/skin/Stylesheet.css" />

<!-- The portlet skins -->
<link id="web_FooterPortlet" rel="stylesheet" type="text/css" href= "/web/skin/portal/webui/component/UIFooterPortlet/DefaultStylesheet.css" />
<link id="web_NavigationPortlet" rel="stylesheet" type="text/css" href= "/web/skin/portal/webui/component/UINavigationPortlet/DefaultStylesheet.css" />
<link id="web_HomePagePortlet" rel="stylesheet" type="text/css" href= "/portal/templates/skin/webui/component/UIHomePagePortlet/DefaultStylesheet.css" />
<link id="web_BannerPortlet" rel="stylesheet" type="text/css" href= "/web/skin/portal/webui/component/UIBannerPortlet/DefaultStylesheet.css" />
...
</head>
```

Note

Both window styles and portlet specification CSS classes are included in the portal skin.

The Skin Service

The skin service of GateIn 3.2 manages various types of skins. This service is reponsible for discovering and deploying the skins into the portal.

Skin configuration

GateIn 3.2 automatically discovers web archives that contain a file descriptor for skins (`WEB-INF/gatein-resources.xml`). This file is reponsible for specifying the portal, portlet and window decorators to be deployed into the skin service.

The full schema can be found in the lib directory:
`exo.portal.component.portal.jar/gatein_resources_1_0.xsd`

Here is an example where a skin (MySkin) with its CSS location is defined, and a few window decorator skins are specified:

```
<gatein-resources>
  <portal-skin>
    <skin-name>MySkin</skin-name>
    <css-path>/skin/myskin.css</css-path>
    <overwrite>false</overwrite>
  </portal-skin>
</gatein-resources>

  <!-- window style -->
  <window-style>
    <style-name>MyThemeCategory</style-name>
    <style-theme>
      <theme-name>MyThemeBlue</theme-name>
    </style-theme>
    <style-theme>
      <theme-name>MyThemeRed</theme-name>
    </style-theme>
    ...
```

Resource Request Filter

Because of the Right-To-Left support, all CSS files need to be retrieved through a Servlet filter and the web application needs to be configured to activate this filter. This is already done for the 01eXoResources.war web application which contains the default skin.

Any new web applications containing skinning CSS files will need to have the following added to their `web.xml` :

```
<filter>
      <filter-name>ResourceRequestFilter</filter-name>
      <filter-class>org.exoplatform.portal.application.ResourceRequestFilter</filter-class>
      </filter>

      <filter-mapping>
      <filter-name>ResourceRequestFilter</filter-name>
      <url-pattern>*.css</url-pattern>
      </filter-mapping>
```

Note

The `display-name` element also needs to be specified in the `web.xml` for the skinning service to work properly with the web application.

The Default Skin

The default skin for GateIn 3.2 is located as part of the `01eXoResource.war`. The main files associated with the skin are shown below:

```
WEB-INF/gatein-resources.xml
WEB-INF/web.xml
skin/Stylesheet.css
```

1. gatein-resources.xml: defines the skin setup to use.
2. web.xml: contains the resource filter and has the display-name set.
3. Stylesheet.css: contains the CSS class definitions for this skin.

gatein-resources.xml
: For the default portal skin, this file contains definitions for the portal skin, the window decorations that this skin provides and defines some Javascript resources which are not related to the skin. The default portal skin does not directly define portlet skins which should be provided by the portlets themeselves.

web.xml
: For the default portal skin, the web.xml of eXoResources.war contains a lot of information which is mostly irrelevant to the portal skinning. The areas of interest in this file is the resourcerequestfilter and the fact that the display-name is set.

Stylesheet.css
: This file contains the main stylesheet of the portal skin. The file is the main entry point to the CSS class definitions for the skin. The content of this file is shown below:

```
@import url(DefaultSkin/portal/webui/component/UIPortalApplicationSkin.css);
@import url(DefaultSkin/webui/component/Stylesheet.css);
@import url(PortletThemes/Stylesheet.css);
@import url(Portlet/Stylesheet.css);
```

1. Skin for the main portal page.
2. Skins for various portal components.

3. Window decoration skins.
4. The portlet specificiation CSS classes.

Instead of defining all the CSS classes in this file, you can import other CSS stylesheet files, some of which may also import other CSS stylesheets. The CSS classes are split up between multiple files to make it easier for new skins to reuse parts of the default skin.

To reuse the CSS stylesheet from the default portal skin, you need to refer to the default skin from eXoResources. For example, to include the window decorators from the default skin within a new portal skin, you need to use this import:

```
@import url(/eXoResources/skin/Portlet/Stylesheet.css);
```

Note

When the portal skin is added to the page, it merges all the CSS stylesheets into a single file.

Creating New Skins

Creating a New Portal Skin

A new portal needs to be added to the portal through the skin service. The web application which contains the skin will need to be properly configured for the skin service to discover them. This means that ResourceRequestFilter and gatein-resources.xml should be configured properly.

Portal Skin Configuration

The gatein-resources.xml needs to specify the new portal skin. This includes specifying the name of the new skin, where to locate its CSS stylesheet file and whether to overwrite an existing portal theme with the same name.

```
<gatein-resources>
  <portal-skin>
    <skin-name>MySkin</skin-name>
    <css-path>/skin/myskin.css</css-path>
    <overwrite>false</overwrite>
  </portal-skin>
</gatein-resources>
```

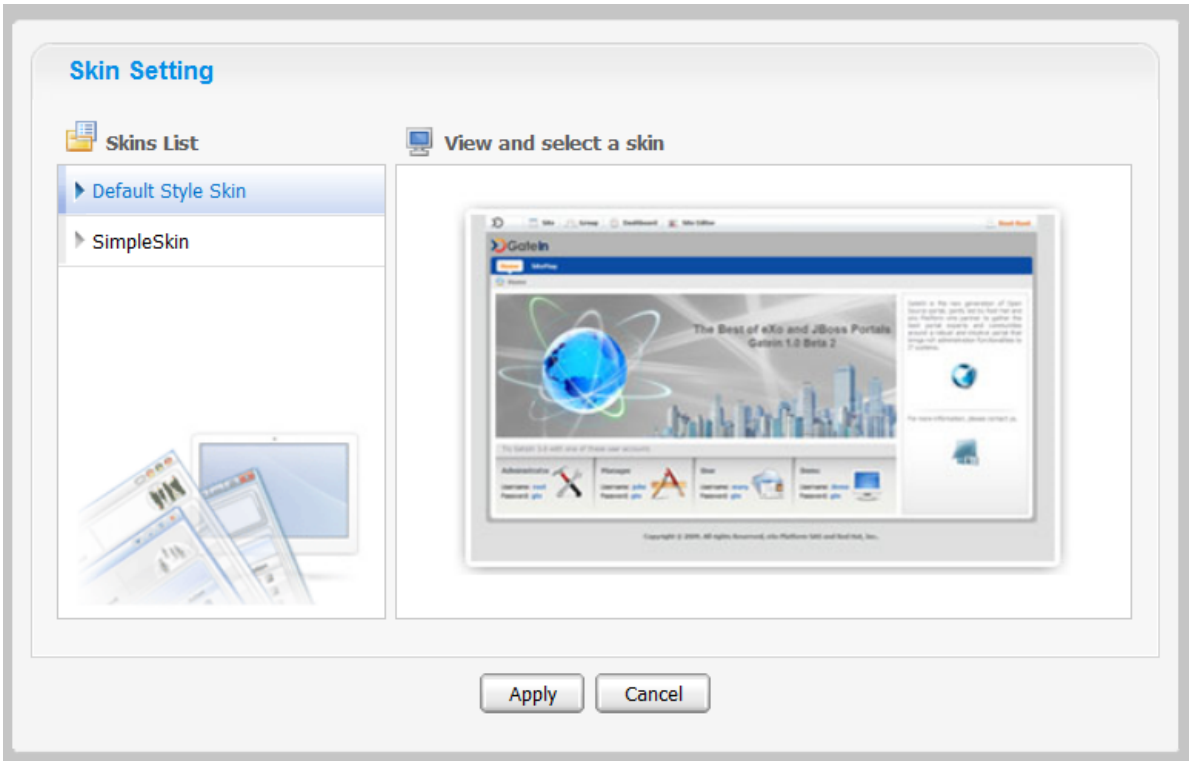
The default portal skin and window styles are defined in `01eXoResources.war/WEB-INF/gatein-resources.xml`.

Note

The CSS for the portal skin needs to contain the CSS for all the window decorators and the portlet specification CSS classes.

Portal Skin Preview Icon

When selecting a skin, it is possible to preview it. The current skin needs to know about the skin icons for all the available skins; otherwise, it will not be able to show the previews. When creating a new portal, it is recommended to include the preview icons of the other skins and to update the other skins with your new portal skin preview.



The portal skin preview icon is specified through the CSS of the portal skin. To display the preview for the current portal skin, you must specify a specific CSS class and set the icon as the background.

For the portal named **MySkin**, the CSS class must be defined as follows:

```
.UIChangeSkinForm .UIItemSelector .TemplateContainer .MySkinImage
```

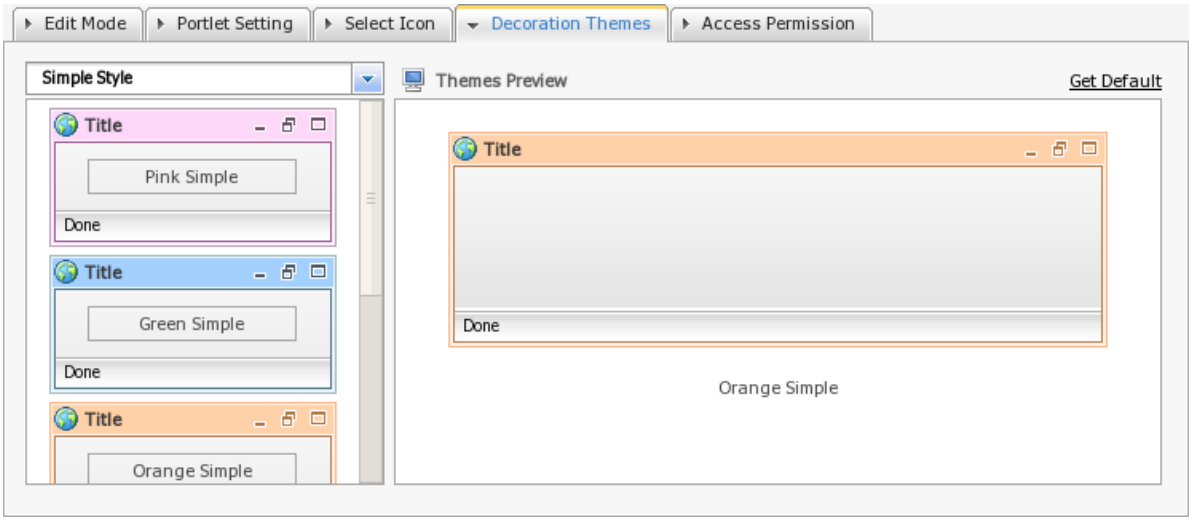
In order for the default skin to know about the skin icon for a new portal skin, the preview screenshot needs to be placed in:
`01eXoResources.war:/skin/DefaultSkin/portal/webui/component/customization/UIChangeSkinForm/background`

The CSS stylesheet for the default portal needs to have the following updated with the preview icon CSS class. For a skin named **MySkin**, you need to update the following: `01eXoResources.war:/skin/DefaultSkin/portal/webui/component/customization/UIChangeSkinForm/Stylesheet.css`

```
.UIChangeSkinForm .UIItemSelector .TemplateContainer .MySkinImage {
  margin: auto;
  width: 329px; height:204px;
  background: url('background/MySkin.jpg') no-repeat top;
  cursor: pointer ;
}
```

Creating a New Window Style

Window styles are the CSS applied to the window decoration. When the administrator selects a new application to add to a page, he can decide which style of decoration would go around the window if any.



Window Style Configuration

Window Styles are defined within the gatein-resources.xml file which is used by the skin service to deploy the window style into the portal. Window styles can belong in with a window style category, this category and the window styles need to be specified in the resources file.

The following gatein-resource.xml fragment adds MyThemeBlue and MyThemeRed to the MyTheme category.

```
<window-style>
  <style-name>MyTheme</style-name>
  <style-theme>
```

```
    <theme-name>MyThemeBlue</theme-name>
  </style-theme>
  <style-theme>
    <theme-name>MyThemeRed</theme-name>
  </style-theme>
</window-style>
```

The windows style configuration for the default skin is configured in:
`01eXoResources.war/WEB-INF/gatein-resources.xml`

> **Note**
>
> When a window style is defined in the gatein-resources.xml file, it will be available to all portlets whether the current portal skin supports the window decorator or not. When a new window decorator is added, it is recommended that you add it to all portal skins or that portal skins share a common stylesheet for window decorators.

Window Style CSS

To display the window decorators for the skin service, it must have the CSS classes with specific naming related to the window style name. The service will try and display the CSS based on this naming. The CSS class must be included as part of the current portal skin for displaying the window decorators.

The location of the window decorator CSS classes for the default portal theme is located at:
`01eXoResources.war/skin/PortletThemes/Stylesheet.css`

Create the CSS file:

```
/*---- MyTheme ----*/
.MyTheme .WindowBarCenter .WindowPortletInfo {
  margin-right: 80px; /* orientation=lt */
  margin-left: 80px; /* orientation=rt */
}
.MyTheme .WindowBarCenter .ControlIcon {
  float: right;/* orientation=lt */
  float: left;/* orientation=rt */
  width: 24px;
  height: 17px;
  cursor: pointer;
  background-image: url('background/MyTheme.png');
}
.MyTheme .ArrowDownIcon {
  background-position: center 20px;
```

```
}
.MyTheme .OverArrowDownIcon {
  background-position: center 116px;
}
.MyTheme .MinimizedIcon {
  background-position: center 44px;
}
.MyTheme .OverMinimizedIcon {
  background-position: center 140px;
}
.MyTheme .MaximizedIcon {
  background-position: center 68px;
}
.MyTheme .OverMaximizedIcon {
  background-position: center 164px;
}
.MyTheme .RestoreIcon {
  background-position: center 92px;
}
.MyTheme .OverRestoreIcon {
  background-position: center 188px;
}
.MyTheme .NormalIcon {
  background-position: center 92px;
}
.MyTheme .OverNormalIcon {
  background-position: center 188px;
}
.UIPageDesktop .MyTheme .ResizeArea {
  float: right;/* orientation=lt */
  float: left;/* orientation=rt */
  width: 18px; height: 18px;
  cursor: nw-resize;
  background: url('background/ResizeArea18x18.gif') no-repeat left top; /* orientation=lt */
  background: url('background/ResizeArea18x18-rt.gif') no-repeat right top; /* orientation=rt */
}
.MyTheme .Information {
  height: 18px; line-height: 18px;
  vertical-align: middle; font-size: 10px;
  padding-left: 5px;/* orientation=lt */
  padding-right: 5px;/* orientation=rt */
  margin-right: 18px;/* orientation=lt */
  margin-left: 18px;/* orientation=rt */
}
```

```
.MyTheme .WindowBarCenter .WindowPortletIcon {
  background-position: left top; /* orientation=lt */
  background-position: right top; /* orientation=rt */
  padding-left: 20px; /* orientation=lt */
  padding-right: 20px; /* orientation=rt */
  height: 16px;
  line-height: 16px;
}
.MyTheme .WindowBarCenter .PortletName {
  font-weight: bold;
  color: #333333;
  overflow: hidden;
  white-space: nowrap;
  width: 100%;
}
.MyTheme .WindowBarLeft {
  padding-left: 12px;
  background-image: url('background/MyTheme.png');
  background-repeat: no-repeat;
  background-position: left -148px;
}
.MyTheme .WindowBarRight {
  padding-right: 11px;
  background-image: url('background/MyTheme.png');
  background-repeat: no-repeat;
  background-position: right -119px;
}
.MyTheme .WindowBarCenter {
  background-image: url('background/MyTheme.png');
  background-repeat: repeat-x;
  background-position: left -90px;
}
.MyTheme .WindowBarCenter .FixHeight {
  height: 21px;
  padding-top: 8px;
}
.MyTheme .MiddleDecoratorLeft {
  padding-left: 12px;
  background: url('background/MyTheme.png') repeat-y left;
}
.MyTheme .MiddleDecoratorRight {
  padding-right: 11px;
  background: url('background/MyTheme.png') repeat-y right;
}
```

```
.MyTheme .MiddleDecoratorCenter {
  background: #ffffff;
}
.MyTheme .BottomDecoratorLeft {
  MyTheme: 12px;
  background-image: url('background/MyTheme.png');
  background-repeat: no-repeat;
  background-position: left -60px;
}
.MyTheme .BottomDecoratorRight {
  padding-right: 11px;
  background-image: url('background/MyTheme.png');
  background-repeat: no-repeat;
  background-position: right -30px;
}
.MyTheme .BottomDecoratorCenter {
  background-image: url('background/MyTheme.png');
  background-repeat: repeat-x;
  background-position: left top;
}
.MyTheme .BottomDecoratorCenter .FixHeight {
  height: 30px;
}
```

How to Set the Default Window Style

To set the default window style to be used for a portal, you need to specify the CSS classes for a theme called `DefaultTheme`.

Note

You do not need to specify the DefaultTheme in gatein-resources.xml.

How to Create New Portlet skins

Portlets often require additional styles that may not be defined by the portal skin. GateIn 3.2 allows you to define additional stylesheets for each portlet and will append the corresponding `link` tags to the `head`.

The link ID will be of the form *`{portletAppName}{PortletName}`*. For example: `ContentPortlet` in `content.war` will give *`id="contentContentPortlet".`*

To define a new CSS file to include whenever a portlet is available on a portal page, the following fragment needs to be added to gatein-resources.xml.

```
<portlet-skin>
```

```
  <application-name>portletAppName</application-name>
  <portlet-name>PortletName</portlet-name>
  <skin-name>Default</skin-name>
  <css-path>/skin/DefaultStylesheet.css</css-path>
</portlet-skin>

<portlet-skin>
  <application-name>portletAppName</application-name>
  <portlet-name>PortletName</portlet-name>
  <skin-name>OtherSkin</skin-name>
  <css-path>/skin/OtherSkinStylesheet.css</css-path>
</portlet-skin>
```

This will load the DefaultStylesheet.css or OtherSkinStylesheet.css when the Default skin or OtherSkin is used respectively.

Note

If the current portal skin is not defined as part of the supported skins, the portlet CSS class will not be loaded. It is recommended that you update portlet skins whenever a new portal skin is created.

Change portlet icons

Each portlet can be represented by a unique icon that you can see in the portlet registry or page editor. This icon can be changed by adding an image to the directory of the portlet webapplication:

- `skin/DefaultSkin/portletIcons/`*`icon_name.png`*.

To use the icon correctly, it must be named after the portlet.

For example, the icon for an account portlet named AccountPortlet would be located at:

- `skin/DefaultSkin/portletIcons/AccountPortlet.png`

Note

You must use `skin/DefaultSkin/portletIcons/` for the directory to store the portlet icon regardless of what skin is going to be used.

How to create a new Portlet Specification CSS Classes

The portlet specification defines a set of default CSS classes that should be available for portlets. These classes are included as part of the portal skin. Please see the portlet specification for a list of the default classes that should be available.

For the default portal skin, the portlet specification CSS classes are defined in: `eXoResources.war/skin/Portlet/Stylesheet.css`

Tips and Tricks

Easier CSS debugging

By default, CSS files are cached and their imports are merged into a single CSS file at the server side. This reduces the number of HTTP requests from the browser to the server.

The optimization code is quite simple as all the CSS files are parsed at the server startup time and all the @import and url(...) references are rewritten to support a single flat file. The result is stored in a cache directly used from the `ResourceRequestFilter`.

Although the optimization is useful for production environments, it may be easier to deactivate this optimization while debugging stylesheets. To do so, set the java system property `exo.product.developing` to `true`.

For example, the property can be passed as a JVM parameter with the `-D` option when running GateIn.

```
sh $JBOSS_HOME/bin/run.sh -Dexo.product.developing=true
```

1. Warning: This option may cause display bugs with certain browsers, such as Internet Explorer.

Some CSS techniques

It is recommended that you have some experiences with CSS before studying GateIn 3.2 CSS.

GateIn 3.2 relies heavily on CSS to create the layout and effects for the UI. Some common techniques for customizing GateIn 3.2 CSS are explained below.

Decorator pattern

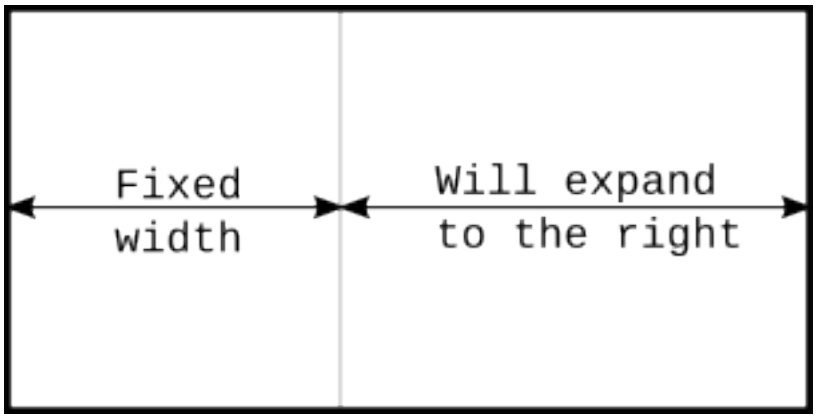
The decorator is a pattern to create a contour or a curve around an area. To achieve this effect, you need to create 9 cells. The BODY is the central area for you to decorate. The other 8 cells are distributed around the BODY cell. You can use the width, height and background image properties to achieve any desired decoration effects.

TopLeft	TopCenter	TopRight
CenterLeft	BODY	CenterRight
BottomLeft	BottomCenter	BottomRight

```
<div class="Parent">
  <div class="TopLeft">
    <div class="TopRight">
      <div class="TopCenter"><span></span></div>
    </div>
  </div>
  <div class="CenterLeft">
    <div class="CenterRight">
      <div class="CenterCenter">BODY</div>
    </div>
  </div>
  <div class="BottomLeft">
    <div class="BottomRight">
      <div class="BottomCenter"><span></span></div>
    </div>
  <div>
</div>
```

Left margin left pattern

Left margin left pattern is a technique to create 2 blocks side by side. The left block has a fixed size and the right block will take the rest of the available space. When the user resizes the browser, the added or removed space will be taken from the right block.




```
<div class="Parent">
  <div style="float: left; width: 100px">
  </div>
  <div style="margin-left: 105px;">
  <div>
  <div style="clear: left"><span></span></div>
</div>
```

Portal Lifecycle

Overview

This chapter describes the portal lifecycle from the application server start to its stop and how requests are handled.

Application Server start and stop

A portal instance is simply a web application deployed as a WAR in an application server. Portlets are also part of an enhanced WAR called a portlet application.

GateIn 3.2 does not require any particular setup for your portlet in most common scenarios and the `web.xml` file can remain without any GateIn 3.2 specific configuration.

During deployment, GateIn 3.2 will automatically and transparently inject a servlet into the portlet application to be able to interact with it. This feature is dependent on the underlying servlet container but will work out of the box on the proposed bundles.

The Command Servlet

The servlet is the main entry point for incoming requests, it also includes some init codes when the portal is launched. This servlet (`org.gatein.wci.command.CommandServlet`) is automatically added during deployment and mapped to `/tomcatgateinservlet`.

This is equivalent to adding the following to `web.xml`.

Note

As the servlet is already configured, this example only aims at providing information.

```
<servlet>
  <servlet-name>TomcatGateInServlet</servlet-name>
  <servlet-class>org.gatein.wci.command.CommandServlet</servlet-class>
  <load-on-startup>0</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>TomcatGateInServlet</servlet-name>
  <url-pattern>/tomcatgateinservlet</url-pattern>
</servlet-mapping>
```

It is possible to filter on the CommandServlet by filtering the URL pattern used by the Servlet mapping.

The example below would create a servlet filter that calculates the time of execution of a portlet request.

The filter class:

```
package org.example;

import java.io.IOException;

import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class MyFilter implements javax.servlet.Filter {

  public void doFilter(ServletRequest request, ServletResponse response,
      FilterChain chain) throws IOException, ServletException
  {
    long beforeTime = System.currentTimeMillis();
    chain.doFilter(request, response);
```

```
    long afterTime = System.currentTimeMillis();
    System.out.println("Time to execute the portlet request (in ms): " + (afterTime - beforeTime));
  }

  public void init(FilterConfig config) throws ServletException
  {
  }

  public void destroy()
  {
  }

}
```

The Java EE web application configuration file (`web.xml`) of the portlet on which we want to know the time to serve a portlet request. As mentioned above, GateIn 3.2 does not require anything special to be included, only the URL pattern to set has to be known.

```
<?xml version="1.0"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.5">

  <filter>
    <filter-name>MyFilter</filter-name>
    <filter-class>org.example.MyFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>MyFilter</filter-name>
    <url-pattern>/tomcatgateinservlet</url-pattern>
    <dispatcher>INCLUDE</dispatcher>
  </filter-mapping>

</web-app>
```

INCLUDE dispatcher

It is important to set `INCLUDE` as a dispatcher as the portal always hits the CommandServlet through a request dispatcher. Without this, the filter will not be triggered, unless the direct access to a resource, such as an image.

Default Portal Configuration

Overview

GateIn 3.2 default homepage URL is `http://{hostname}:{port}/portal/`. There may be multiple independent portals deployed in parallel at any given time, each of which has its root context (i.e.: `http://{hostname}:{port}/sample-portal/`). Each portal is internally composed of one or more 'portals'. It is required to have at least one such portal - the default one is called 'classic'. When accessing the default homepage URL of GateIn 3.2, you are automatically redirected to the 'classic' portal. The default portal performs another important task. When starting up GateIn 3.2 for the first time, its JCR database will be empty (that's where portals keep their runtime-configurable settings). It is the default portal used to detect this, and to trigger automatic data initialization.

Configuration

The following example configuration can be found at: "`02portal.war:/WEB-INF/conf/portal/portal-configuration.xml`".

```
<component>
   <key>org.exoplatform.portal.config.UserPortalConfigService</key>
   <type>org.exoplatform.portal.config.UserPortalConfigService</type>
   <component-plugins>
    <component-plugin>
      <name>new.portal.config.user.listener</name>
      <set-method>initListener</set-method>
      <type>org.exoplatform.portal.config.NewPortalConfigListener</type>
      <description>this listener init the portal configuration</description>
      <init-params>
        <value-param>
          <name>default.portal</name>
          <description>The default portal for checking db is empty or not</description>
          <value>classic</value>
        </value-param>
        ...
      </init-params>
    </component-plugin>
   </component-plugins>
```

```
</component>
```

In this example, the **classic** portal has been set as the default.

Note

Components, component-plugins, and init-params are explained in the Foundations chapter. For now, just note how the `NewPortalConfigListener` component-plugin is used to add configuration to `UserPortalConfigService`, which is designed in this way to allow other components to add configuration to it.

Tips

Delete Portals Definition by Configuration

In some cases, some portal definitions are defined but not used any more. If you want to delete them, you can add some configurations to *portal.war/WEB-INF/conf/portal/portal-configuration.xml*.

To delete a portal definition or a portal template definition, you need to define a component plug-in as the example below:

```
<external-component-plugins>
                <target-component>org.exoplatform.portal.config.UserPortalConfigService</target-component>
   <component-plugin>
    <name>new.portal.config.user.listener</name>
    <set-method>deleteListenerElements</set-method>
    <type>org.exoplatform.portal.config.NewPortalConfigListener</type>
       <description>this listener delete some predefined portal and templates configuration</description>
    <init-params>
     <object-param>
      <name>site.templates.location</name>
      <description>description</description>
      <object type="org.exoplatform.portal.config.SiteConfigTemplates">
       <field name="portalTemplates">
        <collection type="java.util.HashSet">
         <value>
          <string>basic</string>
         </value>
         <value>
          <string>classic</string>
```

```
            </value>
          </collection>
        </field>
      </object>
    </object-param>
    <object-param>
      <name>portal.configuration</name>
      <description>description</description>
      <object type="org.exoplatform.portal.config.NewPortalConfig">
        <field  name="predefinedOwner">
          <collection type="java.util.HashSet">
            <value><string>classic</string></value>
          </collection>
        </field>
        <field name="ownerType"><string>portal</string></field>
      </object>
    </object-param>
  </init-params>
 </component-plugin>
</external-component-plugins>
```

Set the info bar shown by default for portlet

You can set the info bar shown by default for portlets of a portal by adding a property for the *portal-config* configuration in the *portal.xml* file.

```
<properties>
   <entry key="showPortletInfo">1</entry>
</properties>
```

There are two values for "showPortletInfo": 0 and 1. If the value is 1, the info bar of portlets is shown by default. If the value is 0, it is not.

Portal Default Permission Configuration

Overview

The default permission configuration for the portal is defined through `org.exoplatform.portal.config.UserACL` component configuration in the file `02portal.war:/WEB-INF/conf/portal/portal-configuration.xml`.

It defines 8 permissions types:

super.user
: The super-user as *root* has all the rights on the eXo Platform.

portal.administrator.groups
: Any member of those groups are considered administrators. The default value is `/platform/administrators`.

portal.administrator.mstype
: Any user with that membership type would be considered administrator or the associated group with the `manager` by default.

portal.creator.groups
: This list defines all groups that will be able to manage the different portals. Members of this group also have the permission to create new portals. The format is `membership:/group/subgroup`.

navigation.creator.membership.type
: Defines the membership type of group managers. The group managers have the permission to create and edit group pages and they can modify the group navigation.

guests.group
: Any anonymous user automatically becomes a member of this group when they enter the public pages.

mandatory.groups
: Groups that cannot be deleted.

mandatory.mstypes
: Membership types that cannot be deleted.

```
<component>
  <key>org.exoplatform.portal.config.UserACL</key>
  <type>org.exoplatform.portal.config.UserACL</type>
  <init-params>
    <value-param>
      <name>super.user</name>
      <description>administrator</description>
      <value>root</value>
    </value-param>

    <value-param>
      <name>portal.creator.groups</name>
      <description>groups with membership type have permission to manage portal</description>
      <value>*:/platform/administrators,*:/organization/management/executive-board</value>
```

```
    </value-param>

    <value-param>
      <name>navigation.creator.membership.type</name>
          <description>specific membership type have full permission with group navigation</
description>
      <value>manager</value>
    </value-param>
    <value-param>
      <name>guests.group</name>
      <description>guests group</description>
      <value>/platform/guests</value>
    </value-param>
    <value-param>
      <name>access.control.workspace</name>
         <description>groups with memberships that have the right to access the User Control
 Workspace</description>
      <value>*:/platform/administrators,*:/organization/management/executive-board</value>
    </value-param>
  </init-params>
</component>
```

Overwrite Portal Default Permissions

When creating the custom portals and portal extensions, it is possible to override the default configuration by using `org.exoplatform.portal.config.PortalACLPlugin`, configuring it as an external-plugin of `org.exoplatform.portal.config.UserACL` service:

```
<external-component-plugins>
    <target-component>org.exoplatform.portal.config.UserACL</target-component>
    <component-plugin>
      <name>addPortalACLPlugin</name>
      <set-method>addPortalACLPlugin</set-method>
      <type>org.exoplatform.portal.config.PortalACLPlugin</type>
      <description>setting some permission for portal</description>
      <init-params>
        <values-param>
          <name>access.control.workspace.roles</name>
          <value>*:/platform/administrators</value>
          <value>*:/organization/management/executive-board</value>
        </values-param>
        <values-param>
          <name>portal.creation.roles</name>
```

```
        <value>*:/platform/administrators</value>
        <value>*:/organization/management/executive-board</value>
      </values-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

Portal Navigation Configuration

Overview

There are three navigation types available to portal users:

- *the section called “Portal Navigation”*
- *the section called “Group Navigation”*
- *the section called “User Navigation”*

These navigations are configured using the standard XML syntax in the file; "`02portal.war:/WEB-INF/conf/portal/portal-configuration.xml`".

```
<component>
  <key>org.exoplatform.portal.config.UserPortalConfigService</key>
  <type>org.exoplatform.portal.config.UserPortalConfigService</type>
  <component-plugins>
    <component-plugin>
      <name>new.portal.config.user.listener</name>
      <set-method>initListener</set-method>
      <type>org.exoplatform.portal.config.NewPortalConfigListener
    </type>
      <description>this listener init the portal configuration
    </description>
      <init-params>
        <value-param>
          <name>default.portal</name>
          <description>The default portal for checking db is empty or not</description>
          <value>classic</value>
        </value-param>
        <value-param>
          <name>page.templates.location</name>
          <description>the path to the location that contains Page templates</description>
          <value>war:/conf/portal/template/pages</value>
        </value-param>
        <value-param>
```

```
        <name>override</name>
        <description>The flag parameter to decide if portal metadata is overriden on restarting
server
      </description>
        <value>false</value>
      </value-param>
      <object-param>
        <name>site.templates.location</name>
        <description>description</description>
        <object type="org.exoplatform.portal.config.SiteConfigTemplates">
          <field name="location">
            <string>war:/conf/portal</string>
          </field>
          <field name="portalTemplates">
            <collection type="java.util.HashSet">
              <value><string>basic</string></value>
              <value><string>classic</string></value>
            </collection>
          </field>
          <field name="groupTemplates">
            <collection type="java.util.HashSet">
              <value><string>group</string></value>
            </collection>
          </field>
          <field name="userTemplates">
            <collection type="java.util.HashSet">
              <value><string>user</string></value>
            </collection>
          </field>
        </object>
      </object-param>
      <object-param>
        <name>portal.configuration</name>
        <description>description</description>
        <object type="org.exoplatform.portal.config.NewPortalConfig">
          <field name="predefinedOwner">
            <collection type="java.util.HashSet">
              <value><string>classic</string></value>
            </collection>
          </field>
          <field name="ownerType">
            <string>portal</string>
          </field>
          <field name="templateLocation">
```

```
            <string>war:/conf/portal/</string>
          </field>
          <field name="importMode">
            <string>conserve</string>
          </field>
        </object>
      </object-param>
      <object-param>
        <name>group.configuration</name>
        <description>description</description>
        <object type="org.exoplatform.portal.config.NewPortalConfig">
          <field name="predefinedOwner">
            <collection type="java.util.HashSet">
              <value><string>/platform/administrators</string></value>
              <value><string>/platform/users</string></value>
              <value><string>/platform/guests</string></value>
              <value><string>/organization/management/executive-board</string></value>
            </collection>
          </field>
          <field name="ownerType">
            <string>group</string>
          </field>
          <field name="templateLocation">
            <string>war:/conf/portal</string>
          </field>
          <field name="importMode">
            <string>conserve</string>
          </field>
        </object>
      </object-param>
      <object-param>
        <name>user.configuration</name>
        <description>description</description>
        <object type="org.exoplatform.portal.config.NewPortalConfig">
          <field name="predefinedOwner">
            <collection type="java.util.HashSet">
              <value><string>root</string></value>
              <value><string>john</string></value>
              <value><string>mary</string></value>
              <value><string>demo</string></value>
              <value><string>user</string></value>
            </collection>
          </field>
          <field name="ownerType">
```

```
            <string>user</string>
          </field>
          <field name="templateLocation">
            <string>war:/conf/portal</string>
          </field>
          <field name="importMode">
            <string>conserve</string>
          </field>
        </object>
      </object-param>
    </init-params>
  </component-plugin>
</component-plugins>
</component>
```

This XML configuration defines where in the portal's war to look for configuration, and which portals, groups, and user specific views to include in `portal/group/user` navigation. Those files will be used to create an initial navigation when the portal is launched in the first time. That information will then be stored in the JCR content repository, and can then be modified and managed from the portal UI.

Each portal, groups and users navigation is indicated by a configuration paragraph, for example:

```
<object-param>
  <name>portal.configuration</name>
  <description>description</description>
  <object type="org.exoplatform.portal.config.NewPortalConfig">
    <field name="predefinedOwner">
      <collection type="java.util.HashSet">
        <value><string>classic</string></value>
      </collection>
    </field>
    <field name="ownerType">
      <string>portal</string>
    </field>
    <field name="templateLocation">
      <string>war:/conf/portal/</string>
    </field>
    <field name="importMode">
      <string>conserve</string>
    </field>
  </object>
</object-param>
```

1. *predefinedOwner* define the navigation owner, portal will look for the configuration files in folder with this name, if there is no suiable folder, a default portal will be created with name is this value.
2. *ownerType* define the type of portal navigation. It may be a portal, group or user
3. *templateLocation* the classpath where contains all portal configuration files
4. *importMode* The mode for navigation import. There are 4 types of import mode:

    - *conserve*: Import data when it does not exist, otherwise do nothing.
    - *insert*: Import data when it does not exist, otherwise performs a strategy that adds new data only.
    - *merge*: Import data when it does not exist, update data when it exists.
    - *rewrite*: Overwrite data whatsoever.

Base on these parameters, portal will look for the configuration files and create a relevant portal navigation, pages and data import strategy. The portal configuration files will be stored in folders with path look like `{templateLocation}/{ownerType}/{predefinedOwner}`, all navigations are defined in the `navigation.xml` file, pages are defined in pages.xml and portal configuration is defined in `{ownerType}.xml`. For example, with the above configuration, prtal will look for all configuration files from `war:/conf/portal/portal/classic path.`

Portal Navigation

The portal navigation incorporates the pages that can be accessed even when the user is not logged in assuming the applicable permissions allow the public access). For example, several portal navigations are used when a company owns multiple trademarks, and sets up a website for each of them.

The **classic** portal is configured by four XML files in the `02portal.war:/WEB-INF/conf/portal/portal/classic` directory:

portal.xml
: This file describes the layout and portlets that will be shown on all pages. The layout usually contains the banner, footer, menu and breadcrumbs portlets. GateIn 3.2 is extremely configurable as every view element (even the banner and footer) is a portlet.

```
<portal-config
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_objects_1_0     http://
www.gatein.org/xml/ns/gatein_objects_1_0"
  xmlns="http://www.gatein.org/xml/ns/gatein_objects_1_0">
  <portal-name>classic</portal-name>
  <locale>en</locale>
```

```
<access-permissions>Everyone</access-permissions>
<edit-permission>*:/platform/administrators</edit-permission>
<properties>
  <entry key="sessionAlive">onDemand</entry>
  <entry key="showPortletInfo">1</entry>
</properties>

<portal-layout>
  <portlet-application>
    <portlet>
      <application-ref>web</application-ref>
      <portlet-ref>BannerPortlet</portlet-ref>
      <preferences>
        <preference>
          <name>template</name>
          <value>par:/groovy/groovy/webui/component/UIBannerPortlet.gtmpl</value>
          <read-only>false</read-only>
        </preference>
      </preferences>
    </portlet>
    <access-permissions>Everyone</access-permissions>
    <show-info-bar>false</show-info-bar>
  </portlet-application>

  <portlet-application>
    <portlet>
      <application-ref>web</application-ref>
      <portlet-ref>NavigationPortlet</portlet-ref>
    </portlet>
    <access-permissions>Everyone</access-permissions>
    <show-info-bar>false</show-info-bar>
  </portlet-application>

  <portlet-application>
    <portlet>
      <application-ref>web</application-ref>
      <portlet-ref>BreadcumbsPortlet</portlet-ref>
    </portlet>
    <access-permissions>Everyone</access-permissions>
    <show-info-bar>false</show-info-bar>
  </portlet-application>

  <page-body> </page-body>
```

```
      <portlet-application>
        <portlet>
          <application-ref>web</application-ref>
          <portlet-ref>FooterPortlet</portlet-ref>
          <preferences>
            <preference>
              <name>template</name>
              <value>par:/groovy/groovy/webui/component/UIFooterPortlet.gtmpl</value>
              <read-only>false</read-only>
            </preference>
          </preferences>
        </portlet>
        <access-permissions>Everyone</access-permissions>
        <show-info-bar>false</show-info-bar>
      </portlet-application>

    </portal-layout>

  </portal-config>
```

It is also possible to apply a nested container that can also contain portlets. Row, column or tab containers are then responsible for the layout of their child portlets.

Each application references a portlet using the id `portal#{portalName}:/{portletWarName}/{portletName}/{uniqueId}`

Use the `page-body` tag to define where GateIn 3.2 should render the current page.

The defined **classic** portal is accessible to "Everyone" (at `/portal/public/classic`) but only members of the group `/platform/administrators` can edit it.

navigation.xml

This file defines all the navigation nodes of the portal. The syntax is simple using the nested node tags. Each node refers to a page defined in the `pages.xml` file (explained next).

If the administrator want to create node labels for each language, they will have to use `xml:lang` attribute in the label tag with value of `xml:lang` is the relevant locale.

Otherwise, if they want the node label is localized by resource bundle files, the `#{...}` syntax will be used, the enclosed property name serves as a key that is automatically passed to the internationalization mechanism. Thus the literal property name is replaced by a localized value taken from the associated properties file matching the current locale.

For example:

```
<node-navigation
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_objects_1_2     http://
www.gatein.org/xml/ns/gatein_objects_1_2"
  xmlns="http://www.gatein.org/xml/ns/gatein_objects_1_2">
  <priority>1</priority>
  <page-nodes>
    <node>
      <name>home</name>
      <label xml:lang="en">Home</label>
      <page-reference>portal::classic::homepage</page-reference>
    </node>
    <node>
      <name>sitemap</name>
      <label xml:lang="en">SiteMap</label>
      <visibility>DISPLAYED</visibility>
      <page-reference>portal::classic::sitemap</page-reference>
    </node>
    .....
  </page-nodes>
</node-navigation>
```

This navigation tree can have multiple views inside portlets (such as the breadcrumbs portlet) that render the current view node, the site map or the menu portlets.

pages.xml

This configuration file structure is very similar to `portal.xml` and it can also contain container tags. Each application can decide whether to render the portlet border, the window state, the icons or portlet's mode.

```
<page-set
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_objects_1_0     http://
www.gatein.org/xml/ns/gatein_objects_1_0"
  xmlns="http://www.gatein.org/xml/ns/gatein_objects_1_0">

  <page>
    <name>homepage</name>
    <title>Home Page</title>
    <access-permissions>Everyone</access-permissions>
    <edit-permission>*:/platform/administrators</edit-permission>
    <portlet-application>
      <portlet>
        <application-ref>web</application-ref>
```

```
          <portlet-ref>HomePagePortlet</portlet-ref>
          <preferences>
            <preference>
              <name>template</name>
              <value>system:/templates/groovy/webui/component/UIHomePagePortlet.gtmpl</
value>
              <read-only>false</read-only>
            </preference>
          </preferences>
        </portlet>
        <title>Home Page portlet</title>
        <access-permissions>Everyone</access-permissions>
        <show-info-bar>false</show-info-bar>
        <show-application-state>false</show-application-state>
        <show-application-mode>false</show-application-mode>
      </portlet-application>
    </page>

    <page>
      <name>sitemap</name>
      <title>Site Map</title>
      <access-permissions>Everyone</access-permissions>
      <edit-permission>*:/platform/administrators</edit-permission>
      <portlet-application>
        <portlet>
          <application-ref>web</application-ref>
          <portlet-ref>SiteMapPortlet</portlet-ref>
        </portlet>
        <title>SiteMap</title>
        <access-permissions>Everyone</access-permissions>
        <show-info-bar>false</show-info-bar>
      </portlet-application>
    </page>
    .......
</page-set>
```

Group Navigation

Group navigations are dynamically added to the user navigation at login. This allows users to see the menu of all pages assigned to any groups they belong to.

The group navigation menu is configured by three XML files (`navigation.xml`, `pages.xml` and `portlet-preferences.xml`). The syntax used in these files is the same as those covered in *the section called "Portal Navigation"*.

They are also located in the `{templateLocation}/{ownerType}/{predefinedOwner}` directory with `ownerType` is `group` and `predefinedOwner` is the path to the group. For example, `portal.war/WEB-INF/conf/portal/group/platform/administrators/`.

User Navigation

User navigation is the set of nodes and pages that are owned by the user. They are part of the user's dashboard.

Three files configure the user navigation (`navigation.xml`, `pages.xml` and `portlet-preferences.xml`). They are located in the `{templateLocation}/{ownerType}/{predefinedOwner}` directory with `ownerType` is `user` and `predefinedOwner` is username that want to create the navigation. For example, if administrator want to create navigation for user `root`, he has to locate the configuration files in `portal.war/WEB-INF/conf/portal/user/root`

Data Import Strategy

Introduction

In the Portal extension mechanism, developers can define an extension that Portal data can be customized by configurations in the extension. There are several cases which an extension developer wants to define how to customize the Portal data, for example modifying, overwriting or just inserting a bit into the data defined by the portal. Therefore, GateIn also defines several modes for each case and the only thing which a developer has to do is to clarify the usecase and reasonably configure extensions.

This section shows you how data are changes in each mode.

Import Mode

In this section, the following modes for the import strategy are introduced:

- `CONSERVE`
- `MERGE`
- `INSERT`
- `OVERWRITE`

Each mode indicates how the Portal data are imported. The import mode value is set whenever NewPortalConfigListener is initiated. If the mode is not set, the default value will be used in this case. The default value is configurable as a UserPortalConfigService initial param. For example, the bellow configuration means that default value is `MERGE`

```
<component>
```

```
  <key>org.exoplatform.portal.config.UserPortalConfigService</key>
  <type>org.exoplatform.portal.config.UserPortalConfigService</type>
  <component-plugins>
  .....
  </component-plugins>
  <init-params>
    <value-param>
      <name>default.import.mode</name>
      <value>merge</value>
    </value-param>
  </init-params>
</component>
```

The way that the import strategy works with the import mode will be clearly demonstrated in next sections for each type of data.

Data Import Strategy

The 'Portal Data' term which has been referred in the previous sections can be classified into three types of object data: Portal Config, Page Data and Navigation Data; each of which has some differences in the import strategy.

Navigation Data

The navigation data import strategy will be processed to the import mode level as the followings:

- `CONSERVE`: If the navigation exists, leave it untouched. Otherwise, import data.
- `INSERT`: Insert the missing description data, but add only new nodes. Other modifications remains untouched.
- `MERGE`: Merge the description data, add missing nodes and update same name nodes.
- `OVERWRITE`: Always destroy the previous data and recreate it.

In the GateIn navigation structure, each navigation can be referred to a tree which each node links to a page content. Each node contains some description data, such as label, icon, page reference, and more. Therefore, GateIn provides a way to insert or merge new data to the initiated navigation tree or a sub-tree.

The merge strategy performs the recursive comparison of child nodes between the existing persistent nodes of a navigation and the transient nodes provided by a descriptor:

1. Start with the root nodes (which is the effective root node or another node if the parent URI is specified).

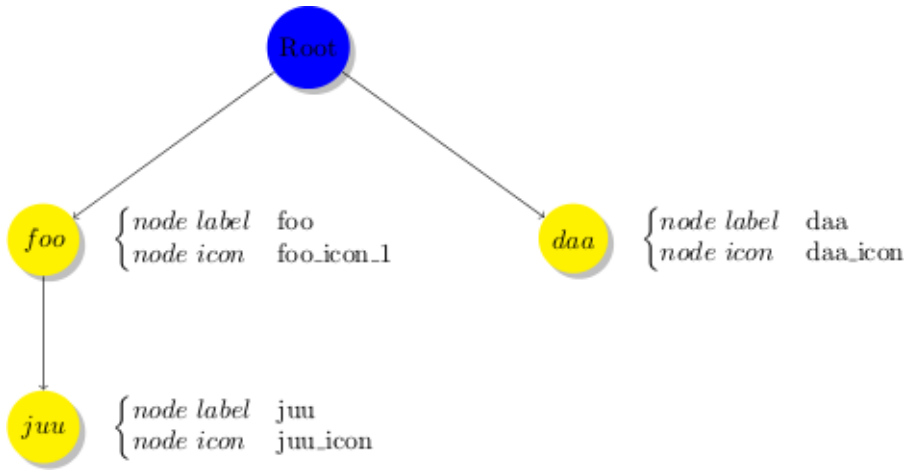
2. Compare the set of child nodes and insert the missing nodes in the persistent nodes.

3. Proceed recursively for each child having the same name.

Let's see the example with two navigation nodes in each import mode. In this case, there are 2 navigation definitions:

```
<node-navigation>
 <page-nodes>
  <node>
   <name>foo</name>
   <icon>foo_icon_1</icon>
   <node>
    <name>juu</name>
    <icon>juu_icon</icon>
   </node>
  </node>
  <node>
   <name>daa</name>
   <icon>daa_icon</icon>
  </node>
 </page-nodes>
</node-navigation>
```

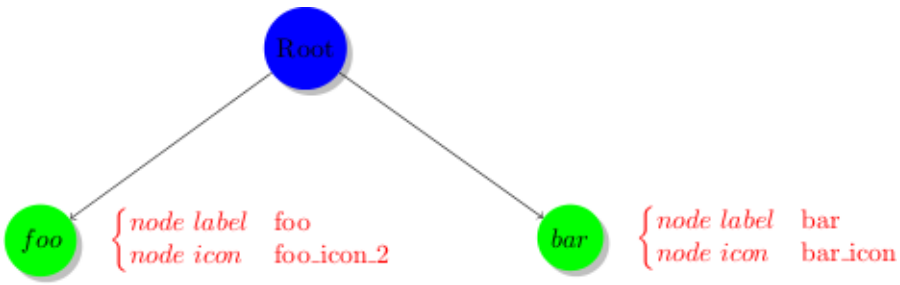
Navigation node tree hierarchy




```
<node-navigation>
 <page-nodes>
  <node>
   <name>foo</name>
   <icon>foo_icon_2</icon>
  </node>
```

```
    <node>
      <name>bar</name>
      <icon>bar_icon</icon>
    </node>
  </page-nodes>
</node-navigation>
```

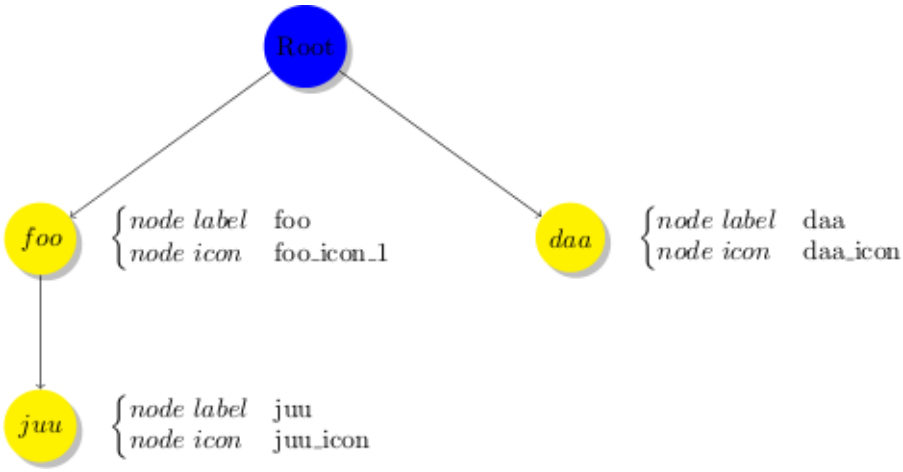
Navigation node tree hierarchy



For example, the *navigation1* is loaded before *navigation2*. The Navigation Importer processes on two navigation definitions, depending on the Import Mode defined in portal configuration.

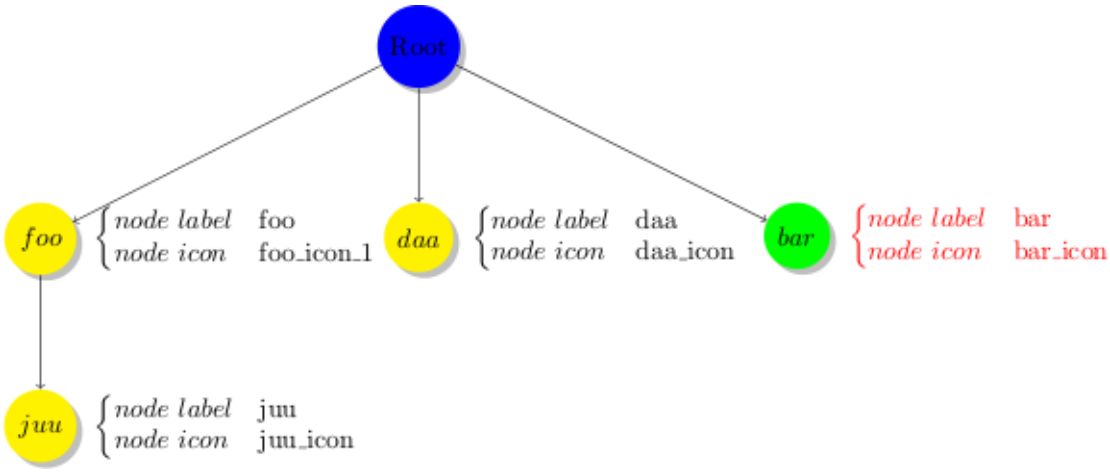
- Case 1: Import mode is `CONSERVE`.

  With the `CONSERVE` mode, data are only imported when they do not exist. So, if the navigation has been created by the *navigation1* definition, the *navigation2* definition does not affect anything on it. We have the result as following



- Case 2: Import mode is `INSERT`.

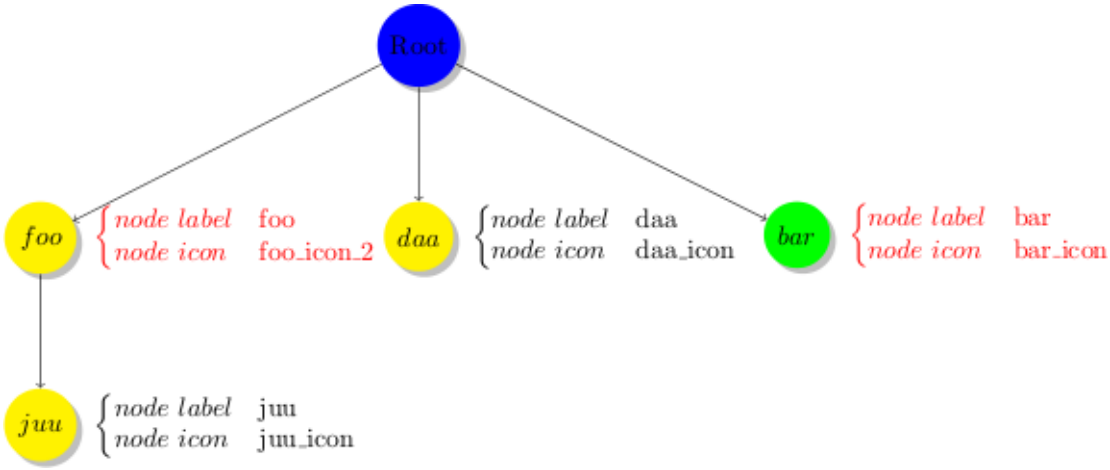
  If a node does not exist, the importer will add new nodes to the navigation tree. You will see the following result:



Hereafter, the node 'bar' is added to the navigation tree, because it does not exist in the initiated data. Other nodes are kept in the import process.

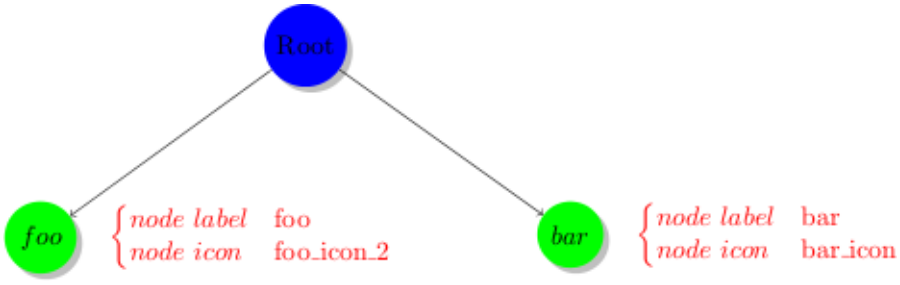
- Case 3: Import mode is `MERGE`.

  The `MERGE` mode indicates that a new node is added to the navigation tree, and updates the node data (such node label and node icon in the example) if it exists.



- Case 4: Import mode is `OVERWRITE`.

  Everything will be destroyed and replaced with new data if the `OVERWRITE` mode is used.



Portal Config

PortalConfig defines the portal name, permission, layout and some properties of a site. These information are configured in the *portal.xml*, *group.xml* or *user.xml*, depending on the site type. The PortalConfig importer performs a strategy that is based on the mode defined in NewPortalConfigListener, including `CONSERVE`, `INSERT`, `MERGE` or `OVERWRITE`. Let's see how the import mode affects in the process of portal data performance:

- `CONSERVE`: There is nothing to be imported. The existing data will be kept without any changes.
- `INSERT`: When the portal config does not exist, create the new portal defined by the portal config definition. Otherwise, do nothing.
- `MERGE` and `OVERWRITE` have the same behavior. The new portal config will be created if it does not exist or update portal properties defined by the portal config definition.

Page Data

The import mode affects the page data import as the same as Portal Config.

Note

If the Import mode is `CONSERVE` or `INSERT`, the data import strategy always performs as the `MERGE` mode in the first data initialization of the Portal.

Internationalization Configuration

Overview

Assumed Knowledge

GateIn 3.2 is fully configurable for internationalization; however, users should have a general knowledge of Internationalization in Java products before attempting these configurations.

Sun Java hosts a comprehensive guide to internationalize Java products at *http://java.sun.com/docs/books/tutorial/i18n/TOC.html*.

All GateIn 3.2 applications contain property files for various languages. They are packaged with the portlets applications in a `WEB-INF/classes/locale/` directory.

These files are located in the `classes` folder of the WEB-INF directory to be loaded by the ClassLoader.

All resource files are in a subfolder named `locale`.

For example, the translations for the *NavigationPortlet* are located in `web.war/WEB-INF/classes/locale/portlet/portal`.

```
NavigationPortlet_de.properties
NavigationPortlet_en.properties
NavigationPortlet_es.properties
NavigationPortlet_fr.properties
NavigationPortlet_nl.properties
NavigationPortlet_ru.properties
NavigationPortlet_uk.properties
NavigationPortlet_ar.xml
```

Those files contain typical `key=value` Java EE properties. For example, the French one:

```
javax.portlet.title=Portlet Navigation
```

There are also properties files in the portal itself. They form the **portal resource bundle**.

From a portlet, you can then access translations from the portlet itself or shared at the portal level, both are aggregated when you need them.

Translation in XML format

It is also possible to use a proprietary XML format to define translations. This is a more convenient way to translate a document for some languages, such as Japanese, Arabic or Russian. Property files have to be ASCII encoded, while the XML file can define its encoding. As a result, it is easier for you to read or edit a translation in XML instead of having to decode and encode the property file.

For more information, refer to *the section called "XML Resources Bundles"*.

Locales configuration

Various languages are available in the portal package. The configuration below will define which languages shown in the "Change Language" section and made available to users.

The `02portal.war:/WEB-INF/conf/common/common-configuration.xml` file of your installation contains the following section:

```
<component>
  <key>org.exoplatform.services.resources.LocaleConfigService</key>
  <type>org.exoplatform.services.resources.impl.LocaleConfigServiceImpl</type>
  <init-params>
    <value-param>
```

```
    <name>locale.config.file</name>
    <value>war:/conf/common/locales-config.xml</value>
   </value-param>
  </init-params>
</component>
```

This configuration points to the locale configuration file.

The locale configuration file (`02portal.war:/WEB-INF/conf/common/locales-config.xml`) contains the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<locales-config>
  <locale-config>
    <locale>en</locale>
    <output-encoding>UTF-8</output-encoding>
    <input-encoding>UTF-8</input-encoding>
    <description>Default configuration for english locale</description>
  </locale-config>

  <locale-config>
    <locale>fr</locale>
    <output-encoding>UTF-8</output-encoding>
    <input-encoding>UTF-8</input-encoding>
    <description>Default configuration for the french locale</description>
  </locale-config>

  <locale-config>
    <locale>ar</locale>
    <output-encoding>UTF-8</output-encoding>
    <input-encoding>UTF-8</input-encoding>
    <description>Default configuration for the arabic locale</description>
    <orientation>rt</orientation>
  </locale-config>
</locales-config>
```

1. *locale* The locale has to be defined, such as *http://ftp.ics.uci.edu-pub-ietf-http-related-iso639.txt*. In this example, "*ar*" is Arabic.
2. *output-encoding* deals with the character encoding. It is recommended that **UTF-8** be used.
3. *input-encoding* In the Java implementation, the encoding parameters will be used for the request response stream. The input-encoding parameter will be used for requesting setCharacterEncoding(..).

4 *description* Description for the language

5 *orientation* The default orientation of text and images is Left-To-Right. GateIn 3.2 supports **Right-To-Left** orientation. Modifying the text orientation is explained in *the section called "RTL (Right To Left) Framework"*.

ResourceBundleService

The resource bundle service is configured in: `02portal.war:/WEB-INF/conf/common/common-configuration.xml`:

```
<component>
  <key>org.exoplatform.services.resources.ResourceBundleService</key>
  <type>org.exoplatform.services.resources.impl.SimpleResourceBundleService</type>
  <init-params>
    <values-param>
      <name>classpath.resources</name>
      <description>The resources  that start with the following package name should be load from
 file system</description>
      <value>locale.portlet</value>
    </values-param>
    <values-param>
      <name>init.resources</name>
      <description>Initiate the following resources during the first launch</description>
      <value>locale.portal.expression</value>
      <value>locale.portal.services</value>
      <value>locale.portal.webui</value>
      <value>locale.portal.custom</value>
      <value>locale.navigation.portal.classic</value>
      <value>locale.navigation.group.platform.administrators</value>
      <value>locale.navigation.group.platform.users</value>
      <value>locale.navigation.group.platform.guests</value>
      <value>locale.navigation.group.organization.management.executive-board</value>
    </values-param>
    <values-param>
      <name>portal.resource.names</name>
      <description>The properties files of  the portal ,  those file will be merged
        into one ResoruceBundle properties </description>
      <value>locale.portal.expression</value>
      <value>locale.portal.services</value>
      <value>locale.portal.webui</value>
      <value>locale.portal.custom</value>
    </values-param>
  </init-params>
```

```
</component>
```

1. *classpath.resources* are discussed in the later section.
2. *init.resources* initiates resources related to portal, group, user resource bundle.
3. *portal.resource.names* defines all resources that belong to the *Portal Resource Bundle*.

   These resources are merged into a single resource bundle which is accessible from anywhere in GateIn 3.2. All these keys are located in the same bundle, which is separated from the navigation resource bundles.

Navigation Resource Bundles

There is a resource bundle for each navigation. A navigation can exist for user, groups, and portal.

The previous example shows bundle definitions for the navigation of the classic portal and of four different groups. Each of these resource bundles occupies a different sphere, they are independent of each other and they are not included in the `portal.resource.names` parameter.

The properties for a group must be in the `WEB-INF/classes/locale/navigation/group/` folder. `/WEB-INF/classes/locale/navigation/group/organization/management/executive-board_en.properties`, for example.

The folder and file names must correspond to the group hierarchy. The group name "`executive-board`" is followed by the iso 639 code.

For each language defined in `LocalesConfig` must have a resource file defined. If the name of a group is changed the name of the folder and/or files of the correspondent navigation resource bundles must also be changed.

Content of `executive-board_en.properties`:

```
organization.title=Organization
organization.newstaff=New Staff
organization.management=Management
```

This resource bundle is only accessible for the navigation of the `organization.management.executive-board` group.

Portlets

Portlets are independent applications and deliver their own resource files.

All shipped portlet resources are located in the **locale/portlet** subfolder. The ResourceBundleService parameter **classpath.resources** defines this subfolder.

Procedure 3.1. Example

To add a Spanish translation to the *GadgetPortlet*.

1. Create the file `GadgetPortlet_es.properties` in: `WEB-INF/classes/locale/portlet/gadget/GadgetPortlet`.
2. In `portlet.xml`, add *Spanish* as a **supported-locale** ('es' is the 2 letters code for Spanish), the **resource-bundle** is already declared and is the same for all languages:

```
<supported-locale>en</supported-locale>
<supported-locale>es</supported-locale>
<resource-bundle>locale.portlet.gadget.GadgetPortlet</resource-bundle>
```

See the portlet specification for more details about the portlet internationalization.

Standard portlet resource keys

The portlet specifications define three standard keys: Title, Short Title and Keywords. Keywords are formatted as a comma-separated list of tags.

```
javax.portlet.title=Breadcrumbs Portlet
javax.portlet.short-title=Breadcrumbs
javax.portlet.keywords=Breadcrumbs, Breadcrumb
```

Debugging resource bundle usage

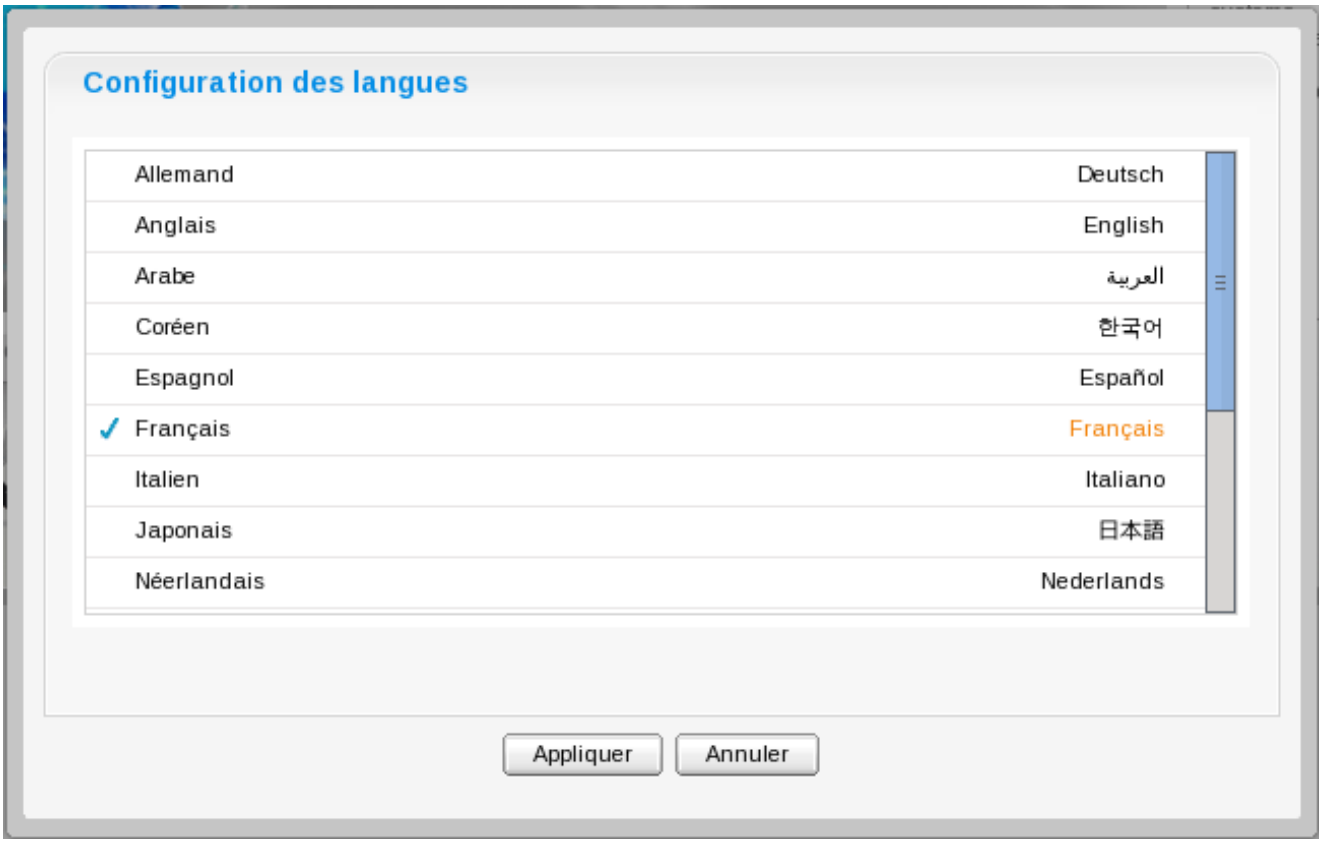
When translating an application, it can sometimes be difficult to find the right key for a given property.

Execute the portal in the **debug mode** and select the special language from the available languages,; **Magic locale**.

This feature translates a key to the same key value.

For example, the translated value for the key "*organization.title*" is simply the value "*organization.title*". Selecting that language allows use of the portal and its applications with all the keys visible. This makes it easier to find out the correct key for a given label in the portal page.

Translating the language selection form



When choosing a language as on the screenshot above, the user is presented with a list of languages on the left side in the current chosen language and on the right side, the same language translated into its own language. Those texts are obtained from the JDK API `java.util.Locale.getDisplayedLanguage()` and `java.util.Locale.getDisplayedCountry()` (if needed) and all languages may not be translated and can also depend on the JVM currently used. It is still possible to override those values by editing the `locale.portal.webui` resource bundle, to do so edit the file `gatein.ear/02portal.war/WEB-INF/classes/locale/portal/webui_xx_yy.properties` where `xx_yy` represents the country code of the language in which you want to translate a particular language. In that file, add or modify a key such as `Locale.xx_yy` with the value being the translated string.

Example 3.1. Changing the displayed text for Traditional Chinese in French

First edit `gatein.ear/02portal.war/WEB-INF/classes/locale/portal/webui_fr.properties` where `ne` is the country code for French, and add the following key into it:

```
Locale.zh_TW=Chinois traditionnel
```

After a restart the language will be updated in the user interface when a user is trying to change the current language.

RTL (Right To Left) Framework

The text orientation depends on the current locale setting. The orientation is a Java 5 enum that provides a set of functionalities:

```
LT, // Western Europe
RT, // Middle East (Arabic, Hebrew)
TL, // Japanese, Chinese, Korean
TR; // Mongolian
public boolean isLT() { ... }
public boolean isRT() { ... }
public boolean isTL() { ... }
public boolean isTR() { ... }
```

The object defining the Orientation for the current request is the `UIPortalApplication`. However, it should be accessed at runtime using the `RequestContext` that delegates to the `UIPortalApplication`.

In case of `PortalRequestContext`, it directly delegates as the `PortalRequestContext` has a reference to the current `UIPortalApplication`.

In case of a different context, such as the `PortletRequestContext`, it delegates to the parent context given the fact that the root `RequestContext` is always a `PortalRequestContext`.

Groovy templates

Orientation is defined by implicit variables in the Groovy binding context:

Orientation
: The current orientation as an Orientation

isLT
: The value of orientation.isLT()

isRT
: The value of orientation.isRT()

dir
: The string 'ltr' if the orientation is LT or the string 'rtl' if the orientation is RT.

Stylesheet

The skin service handles stylesheet rewriting to accommodate the orientation. It works by appending -lt or -rt to the stylesheet name.

For instance: `/web/skin/portal/webui/component/UIFooterPortlet/DefaultStylesheet-rt.css` will return the same stylesheet as `/web/skin/portal/webui/component/UIFooterPortlet/DefaultStylesheet.css` but processed for the RT orientation. The *`-lt`* suffix is optional.

Stylesheet authors can annotate their stylesheet to create content that depends on the orientation.

Example 3.2.

In the example, we need to use the orientation to modify the float attribute that will make the horizontal tabs either float on left or on right:

```
float: left; /* orientation=lt */
float: right; /* orientation=rt */
font-weight: bold;
text-align: center;
white-space: nowrap;
```

The LT produced output will be:

```
float: left; /* orientation=lt */
font-weight: bold;
text-align: center;
white-space: nowrap;
```

The RT produced output will be:

```
float: right; /* orientation=rt */
font-weight: bold;
text-align: center;
white-space: nowrap;
```

Example 3.3.

In this example, you need to modify the padding according to the orientation:

```
color: white;
line-height: 24px;
padding: 0px 5px 0px 0px; /* orientation=lt */
padding: 0px 0px 0px 5px; /* >orientation=rt */
```

The LT produced output will be:

```
color: white;
line-height: 24px;
padding: 0px 5px 0px 0px; /* orientation=lt */
```

The RT produced output will be:

```
color: white;
line-height: 24px;
padding: 0px 0px 0px 5px; /* orientation=rt */
```

Images

Sometimes, it is necessary to create the RT version of an image that will be used from a template or from a stylesheet. However, symmetric images can be automatically generated avoiding the necessity to create a mirrored version of an image and furthermore avoiding maintenance cost.

The web resource filter uses the same naming pattern as the skin service. When an image ends with the -rt suffix, the portal will attempt to locate the original image and create a mirror of it.

For instance: requesting the image `/GateInResources/skin/DefaultSkin/webui/component/UITabSystem/UITabs/background/NormalTabStyle-rt.gif` returns a mirror of the image `/GateInResources/skin/DefaultSkin/webui/component/UITabSystem/UITabs/background/NormalTabStyle.gif`.

Note

It is important to consider whether the image to be mirrored is symmetrical as this will impact its final appearance.

Here is an example combining stylesheet and images:

```
line-height: 24px;
background: url('background/NavigationTab.gif') no-repeat right top; /* orientation=lt */
background: url('background/NavigationTab-rt.gif') no-repeat left top; /* orientation=rt */
padding-right: 2px; /* orientation=lt */
padding-left: 2px; /* orientation=rt */
```

Client side JavaScript

The `eXo.core.I18n` object provides the following parameters for orientation:

getOrientation()
: Return either the string lt or rt

getDir()
: Return either the string ltr or rtl

isLT()
: Return true for LT

isRT()
: Return true of RT

XML Resources Bundles

Motivation

Resource bundles are usually stored in property files. However, as property files are plain files, issues with the encoding of the file may arise. The XML resource bundle format has been developed to provide an alternative to property files.

- The XML format declares the encoding of the file. This avoids use of the native2ascii program which can interfere with encoding.
- Property files generally use the ISO 8859-1 character encoding which does not cover the full unicode charset. As a result, languages, such as Arabic, would not be natively supported.
- Tooling for XML files is better supported than the tooling for Java property files; thus, the XML editor copes well with the file encoding.

XML format

The XML format is very simple and has been developed based on the *DRY* (Don't Repeat Yourself) principle. The resource bundle keys are hierarchically defined and we can leverage the hierarchic

nature of the XML for that purpose. Here is an example of turning a property file into an XML resource bundle file:

```
UIAccountForm.tab.label.AccountInputSet = ...
UIAccountForm.tab.label.UIUserProfileInputSet = ...
UIAccountForm.label.Profile = ...
UIAccountForm.label.HomeInfo= ...
UIAccountForm.label.BusinessInfo= ...
UIAccountForm.label.password= ...
UIAccountForm.label.Confirmpassword= ...
UIAccountForm.label.email= ...
UIAccountForm.action.Reset= ...
```

```
<?xml version="1.0" encoding="UTF-8"?>
<bundle>
 <UIAccountForm>
  <tab>
   <label>
    <AccountInputSet>...</AccountInputSet>
    <UIUserProfileInputSet>...</UIUserProfileInputSet>
   </label>
  </tab>
  <label>
   <Profile>...</Profile>
   <HomeInfo>...</HomeInfo>
   <BusinessInfo>...</BusinessInfo>
   <password>...</password>
   <Confirmpassword>...</Confirmpassword>
   <email>...</email>
  </label>
  <action>
   <Reset>...</Reset>
  </action>
 </UIAccountForm>
</bundle>
```

Portal support

To be loaded by the portal at runtime (actually the resource bundle service), the name of the file must be the same as a property file and it must use the **.xml** suffix.

For example, for the Account Portlet to be displayed in Arabic, the resource bundle would be **AccountPortlet_ar.xml** rather than **AccountPortlet_ar.properties**.

JavaScript Inter Application Communication

Overview

JavaScript Inter Application Communication is designed to allow applications within a page to exchange data. This library is made for broadcasting messages on topic.

It is based on 3 functions:

- Subscribe.
- Publish.
- Unsubscribe.

A subscription to a topic will receive any subtopic messages. For example, the application subscribed to "`/eXo/application`" will receive messages sent on the "`/eXo/application/map`" topic. A message sent on "`/eXo`" would not be received.

Subscription Topics

/eXo
: This topic contains all the events generated by eXo Platform.

/eXo/portal/notification
: A message sent on this topic will prompt a pop-up notification at the top right of the screen.

Library

The Inter Application Communication library is found in `01eXoResources.war:/javascript/eXo/core/Topic.js`

```
/**
 * publish is used to publish an event to the other subscribers to the given channels
 * @param {Object} senderId is a string that identify the sender
 * @param {String} topic is the topic that the message will be published
 * @param {Object} message is the message that's going to be delivered to the subscribers to the topic
 */
Topic.prototype.publish = function(/*Object*/ senderId, /*String*/ topicName, /*Object*/ message ) { ... }

/**
 * isSubscribed is used to check if a function receive the events from a topic
```

```
 * @param {String} topic The topic.
 * @param {Function} func is the name of the function of obj to call when a message is received
 on the topic
 */
Topic.prototype.isSubscribed = function(/*String*/ topic, /*Function*/ func) { ... }

/**
 * subscribe is used to subscribe a callback to a topic
 * @param {String} topic is the topic that will be listened
 * @param {Function} func is the name of the function of obj to call when a message is received
 on the topic
 *
 * func is a function that take a Object in parameter. the event received have this format:
 * {senderId:senderId, message:message, topic: topic}
 *
 */
Topic.prototype.subscribe = function(/*String*/ topic, /*Function*/ func) { ... }

/**
 * unsubscribe is used to unsubscribe a callback to a topic
 * @param {String} topic is the topic
 * @param {Object} id is the id of the listener we want to unsubscribe
 */
Topic.prototype.unsubscribe = function(/*String*/ topic, /*Object*/ id) { ... }

Topic.prototype.initCometdBridge = function() { ... }
```

Syntax

The three messaging functions require particular objects and definitions in their syntax:

Subscribe

The `subscribe` function is used to subscribe a callback to a topic. It uses the following parameters:

topic

The topic that will be listened for.

func

The name of the object function to call when a message is received on the topic. It has to be a function that takes an Object parameter. The event received will have this format:

```
{
  senderId:senderId,
```

```
  message:message,
  topic: topic
}
```

Publish

The `publish` function is used to publish an event to the other subscribered applications through the given channels. Its parameters are:

senderId

This is a string that identifies the sender.

topicName

The topic that the message will be published.

message

This is the message body to be delivered to the subscribers to the topic.

Unsubscribe

The `unsubscribe` function is used to unsubscribe a callback to a topic. The required parameters are:

topic

The topic from which will be unsubscribed.

id

This is the context object.

Example of Javascript events usage

```
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>
<div>
  <p>
    Received messages:
    <div id="received_<portlet:namespace/>">

    </div>
  </p>

  <p>
    Send message:
                    <input type="text" id="msg_<portlet:namespace/>"/> <a href="#"
 onclick="send_<portlet:namespace/>();">send</a>
  </p>
</div>
```

```
<script type="text/javascript">

 Function.prototype.bind = function(object) {
  var method = this;
   return function() {
   method.apply(object, arguments);
  }
 }

 function send_<portlet:namespace/>() {
  var msg = document.getElementById("msg_<portlet:namespace/>").value;
  eXo.core.Topic.publish("<portlet:namespace/>", "/demo", msg);
 }

 function Listener_<portlet:namespace/>(){

 }

 Listener_<portlet:namespace/>.prototype.receiveMsg = function(event) {
  document.getElementById("received_<portlet:namespace/>").innerHTML =
   document.getElementById("received_<portlet:namespace/>").innerHTML + "<br />* " +
   event.senderId + ": " + event.message;
 }

 function init_<portlet:namespace/>() {
  var listener_<portlet:namespace/> = new Listener_<portlet:namespace/>();
                                eXo.core.Topic.subscribe("/demo",          listener_<portlet:namespace/
>.receiveMsg.bind(listener_<portlet:namespace/>));
 }

 init_<portlet:namespace/>();
</script>
```

Upload Component

Upload Service

The service is defined by the class: `org.exoplatform.upload.UploadService`;

This can be configured with the following xml code:

```
<component>
```

```
  <type>org.exoplatform.upload.UploadService</type>
   <init-params>
    <value-param>
     <name>upload.limit.size</name>
     <description>Maximum size of the file to upload in MB</description>
     <value>10</value>
    </value-param>
   </init-params>
  </component>
```

This code allows for a default upload size limit for the service to be configured. The value unit is in MegaBytes.

This limit will be used by default by all applications if no application-specific limit is set. Setting a different limit for applications is discussed in a later section.

If the value is set to `0`, the upload size is unlimited.

Procedure 3.2. How to use the upload component

1. Create an object type `org.exoplatform.webui.form.UIFormUploadInput`.

   Two constructors are available for this:

   ```
   public UIFormUploadInput(String name, String bindingExpression)
   ```

   or:

   ```
   public UIFormUploadInput(String name, String bindingExpression, int limit)
   ```

   This is an example using the second form:

   ```
   PortletRequestContext pcontext =
    (PortletRequestContext)WebuiRequestContext.getCurrentInstance();
   PortletPreferences portletPref = pcontext.getRequest().getPreferences();
   int limitMB = Integer.parseInt(portletPref.getValue("uploadFileSizeLimitMB", "").trim());
   UIFormUploadInput uiInput = new UIFormUploadInput("upload", "upload", limitMB);
   ```

2. To obtain the limit from the `xml` configuration, this piece of code can be added to the either `portlet.xml` or `portlet-preferences.xml`:

```
<preference>
  <name>uploadFileSizeLimitMB</name>
  <value>30</value>
  <read-only>false</read-only>
</preference>
```

Again, the `0` value means an unlimited upload size, and the value unit is set in MegaBytes.

3. Use the `getUploadDataAsStream()` method to get the uploaded data:

```
UIFormUploadInput input = (UIFormUploadInput)uiForm.getUIInput("upload");
InputStream inputStream = input.getUploadDataAsStream();
...
jcrData.setValue(inputStream);
```

4. The upload service stores a temporary file on the file system during the upload process. When the upload is finished, the service must be cleaned to:

    1. Delete the temporary file.
    2. Delete the classes used for the upload.

    Use the`removeUpload()` method defined in the upload service to purge the file:

```
UploadService uploadService = uiForm.getApplicationComponent(UploadService.class) ;
UIFormUploadInput uiChild = uiForm.getChild(UIFormUploadInput.class) ;
uploadService.removeUpload(uiChild.getUploadId()) ;
```

Saving the uploaded file

Ensure the file is saved **before** the service is cleaned.

Deactivation of the Ajax Loading Mask Layer

Purpose

The loading mask layer is deployed after an ajax-call. It aims at blocking the GUI to prevent further user actions until the the ajax-request has been completed.

However, the mask layer may need to be deactivated in instances where the portal requires user instructions before the previous instructions have been carried out.

Procedure 3.3. How to deactivate the ajax-loading mask

1. Generate a script to make an asynchronous ajax-call. Use the `uicomponent.doAsync()` method rather than the `uicomponent.event()` method.

   For example:

```
<a href="<%=uicomponent.doAsync(action, beanId, params)%>" alt="">Asynchronous</a>
```

2. The `doAsync()` method automatically adds the following new parameter to the parameters list; `asyncparam = new Parameter(AJAX ASYNC,"true"); (AJAX ASYNC == "ajax async")`

   This request is asynchronous and the ajax-loading mask will not deployed.

   #### Note

   An asynchronous request can still be made using the `uicomponent.event()`. When using this method, the asyncparam must be added manually.

The GUI will be blocked to ensure that the user can only request one action at one time and while the request seems to be synchronous, all ajax requests are always asynchronous. For further information, refer to *the section called "Synchronous issue"*.

Synchronous issue

Most web browsers support ajax requests in two modes: *Synchronous* and *Asynchronous*. This mode is specified with a boolean `bAsync` parameter.

```
var bAsync = false; // Synchronous
request.open(instance.method, instance.url, bAsync);
```

However, to work with browsers that do not support the *Synchronous* requests, `bAsync` is always set to true (Ajax request will always be asynchronous).

```
// Asynchronous request
request.open(instance.method, instance.url, true);
```

JavaScript Configuration

Managing JavaScript in an application like GateIn 3.2 is a critical part of the configuration work. Configuring the scripts correctly will result in the faster response time from the portal.

Every portlet can have its own JavaScript code but in many cases, it is more convenient to reuse some existing shared libraries. For that reason, GateIn 3.2 has a mechanism to easily register the libraries that will be loaded when every page is rendered.

To do so, every WAR deployed in GateIn 3.2 can register the `.js` files with the **gatein-resources.xml** configuration file.

The example code snippet below is found in the **gatein-resources.xml** in the **eXoResources.war** file.

```
<javascript>
  <param>
    <js-module>eXo</js-module>
    <js-path>/javascript/eXo.js</js-path>
    <js-priority>0</js-priority>
  </param>
</javascript>

<!-- CORE Javascripts -->
<javascript>
  <param>
    <js-module>eXo.core.Utils</js-module>
    <js-path>/javascript/eXo/core/Util.js</js-path>
    <js-priority>1</js-priority>
  </param>
  <param>
    <js-module>eXo.core.DOMUtil</js-module>
    <js-path>/javascript/eXo/core/DOMUtil.js</js-path>
    <js-priority>1</js-priority>
  </param>
  <param>
    <js-module>eXo.core.Browser</js-module>
    <js-path>/javascript/eXo/core/Browser.js</js-path>
    <js-priority>2</js-priority>
  </param>
  <param>
    <js-module>eXo.core.MouseEventManager</js-module>
    <js-path>/javascript/eXo/core/MouseEventManager.js</js-path>
  </param>
  <param>
    <js-module>eXo.core.UIMaskLayer</js-module>
    <js-path>/javascript/eXo/core/UIMaskLayer.js</js-path>
  </param>
  <param>
    <js-module>eXo.core.Skin</js-module>
```

```
    <js-path>/javascript/eXo/core/Skin.js</js-path>
  </param>
  <param>
    <js-module>eXo.core.DragDrop</js-module>
    <js-path>/javascript/eXo/core/DragDrop.js</js-path>
  </param>
  <param>
    <js-module>eXo.core.DragDrop2</js-module>
    <js-path>/javascript/eXo/core/DragDrop2.js</js-path>
  </param>
</javascript>
```

Note that registered JavaScript files will be merged into a single `merged.js` file when the server loads. This reduces the number of HTTP calls as shown in the homepage source code:

```
<script type="text/javascript" src="/portal/javascript/merged.js"></script>
```

Although this optimization is useful for a production environment, it may be easier to deactivate this optimization while debugging JavaScript problems.

To do this, set the Java system property `exo.product.developing` to `true`. GateIn provides two startup scripts that define this property in gatein-dev.sh (for Linux, Mac) and gatein-dev.bat (for Windows).

To generate the `merged.js` file, set this property to `false`. If the property is not set, the default value is false.

The property can be passed as a JVM parameter with the `-D` option in your `GateIn.sh` or `GateIn.bat` startup script.

Every JavaScript file is associated with a module name which acts as a namespace.

Inside the associated JavaScript files, the eXo JavaScript objects are exposed as global variables named after the module.

For example:

```
eXo.core.DragDrop = new DragDrop();
```

It is also possible to use the `eXo.require()`method to lazy load and evaluate some JavaScript codes. This is quite useful for the portlet or gadget applications that will use this JavaScript only

once. Otherwise, if the library is reusable in several places, it is better to define it in the **gatein-resources.xml** file.

Navigation Controller

Description

The navigation controller is a major enhancement of GateIn that has several goals:

- Provide non-ambiguous URLs for resources managed by the portal, such as navigation. Previously, different resources were possible for a single URL, even worse, the set of resources available for an URL depends on private navigations (groups and dashboard).
- Decouple the HTTP request from the portal request. Previously, both were tightly coupled, for instance, the URL for a site had to begin with */public/{sitename}* or */private/{sitename}* .The navigation controller provides a flexible and configurable mapping.
- Provide a more friendly URL and let portal administrator configure how the HTTP request should look like.

Controller in Action

Controller

The `WebAppController` is the component of GateIn that processes HTTP invocations and transforms them into a portal request. It has been improved with the addition of a request mapping engine (**controller**) whose role is to make the HTTP request decouple and create a portal request. The mapping engine makes two essential tasks:

- Create a **Map<QualifiedName, String>** from an incoming HTTP request.
- Render a **Map<QualifiedName, String>** as an HTTP URL.

The goal of the controller (mapping engine) is to **decouple** the request processed by GateIn from the incoming HTTP request. Indeed, a request contain data that determine how the request will be processed and such data can be encoded in various places in the request, such as the request path, or a query parameter. The controller allows GateIn to route a request according to a set of parameters (a map) instead of the servlet request.

The controller configuration is declarative in an .xml file, allowing easy reconfiguration of the routing table and it is processed into an internal data structure that is used to perform resolution (routing or rendering).

Building controller

The controller configuration that contains the routing rules is loaded from the **controller.xml** file that is retrieved in the GateIn configuration directory. Its location is determined by the **gatein.controller.config** property.

WebAppController loads and initializes the mapping engine.

```
<!-- conf/portal/controller-configuration.xml of portal.war -->
<component>
  <type>org.exoplatform.web.WebAppController</type>
    <init-params>
      <value-param>
        <name>controller.config</name>
        <value>${gatein.portal.controller.config}</value>
      </value-param>
    </init-params>
</component>
```

GateIn's extension project can define their own routing table, thanks to the extension mechanism.

The **controller.xml** file can be changed and reloaded at runtime. This helps the test of different configurations easily (configuration loading operations) and provides more insight into the routing engine (the findRoutes operation). See **Rebuiding controller** below for more details.

- **ReBuilding controller**

The WebAppController is annotated with `@Managed` annotations and is bound under the `view=portal,service=controller` JMX name and under the "portalcontroller" REST name.

It provides the following attributes and operations:

- Attribute configurationPath: the "read-only" configuration path of the **controller.xml** file.
- Operation loadConfiguration: load a new configuration file from a specified XML path.
- Operation reloadConfiguration: reload the configuration file.
- Operation findRoutes: route the request argument through the controller and returns a list of all parameter map resolutions. The argument is a request URI, such as */groups/:platform:administrators/administration/registry*. It returns a string representation (`List<Map>`) of the matched routes.

Controller Configuration (controller.xml)

Most of the controller configuration cares about defining rules (Routing table - contains routes object) that will drive the resolution. Routes are processed during the controller initialization to give a tree of node.

- Each node is related to its parent with a matching rule that can either be an **exact string matching** or a **regular expression matching**.

- Each node is associated with a set of parameters.

A parameter is defined by a qualified name and there are three kinds of parameters explained in the sections below.

Route parameters

Route parameters defines a fixed value associate with a qualified name.

- Routing: route parameters allow the controller to distinguish branches easily and route the request accordingly.
- Rendering: the system will select a route to render an URL if all route parameters are always matched.

Example:

```
<route path="/foo">
  <route-param qname="gtn:handler">
    <value>portal</value>
  </route-param>
</route>
```

This configuration matches the request path "**/foo**" to the map (gtn:handler=portal). Conversely, it renders the (gtn:handler=portal) map as the "**/foo**" URL. This example shows two concepts:

- exact path matching ("/foo")
- route parameters ("gtn:handler")

Path parameters - Regular expression support

Path parameters allow to associate a portion of the request path with a parameter. Such parameter will match any non empty portions of text except the **/** character (that is the [^/]+ regular expression) otherwise they can be associated with a regular expression for matching specific patterns. Path parameters are mandatory for matching since they are a part of the request path, however it is allowed to write regular expression matching an empty value.

- Routing: route is accepted if the regular expression is matched.
- Rendering: the system will select a route to render an URL if all route parameters are always matched.

Encoding

Path parameters may contain the '/' character which is a reserved char for the URI path. This case is specially handled by the navigation controller by using a special character to replace the '/' literals. By default, the character is the colon **":"** and can be changed to other possible values (see controller XML schema for possible values) to give a greater amount of flexibility.

This encoding is applied only when the encoding is performed for parameters having a mode set to the `default-form` value, for instance, it does not happen for navigation node URI (for which **/** are encoded literally). The separator escape char can still be used but under it is percent escaped form, so by default, a path parameter value containing the colon **":"** would be encoded as `%3A` and conversely the `%3A` value will be decoded as the colon **":"**.

Example: No pattern is define, the default one [^/]+ will be used:

```
<route path="/{gtn:path}">
</route>
```

As a result of the example above, routing and rendering is as below:

```
Routing and Rendering
Path "/foo"      <--> the map (gtn:path=foo)

Path "/foo:bar"  <--> the map (gtn:path=foo/bar)
```

If the request path contains another "/" char, it will not work. The default encoding mode is **default-form**. In the example above, "/foo/bar" is not matched, so the system returns an empty parameter map.

However, this problem could be solved with the following configuration:

```
<route path="/{gtn:path}">
  <path-param encoding="preserve-path" qname="gtn:path">
    <pattern>.*</pattern>
  </path-param>
</route>
```

- The ".*" declaration allows matching any char sequence.

- The "*preserve-path*" encoding tells the engine that the "/" chars should be handled by the path parameter itself as they have a special meaning for the router. Without this special encoding, "/" would be rendered as the ":" character and conversely the ":" character would be matched as the "/" character.

Request parameters

Request parameters are matched from the request parameters (GET or POST). The match can be optional as their representation in the request allows it.

- **Routing**:
  - Route is accepted when a required parameter is present and matched in the request.
  - Route is accepted when an optional parameter is absent or matched in the request.
- **Rendering**:
  - For required parameters, the system will select a route to render an URL when the parameter is present and matched in the map.
  - For optional parameters, the system will select a route to render an URL when the parameter is absent or matched in the map.

Example:

```
<route path="/">
  <request-param name="path" qname="gtn:path"/>
</route>
```

Request parameters are declared by a `request-param` element and will match any value by default. A request like "/?path=foo" is mapped to the (gtn:path=foo) map. The `name` attribute of the `request-param` tag defines the request parameter value. This element accepts more configuration:

- A `value` or a `pattern` element that is a child element used to match a constant or a pattern.
- A `control-mode` attribute with the `optional` or `required` value indicates if matching is mandatory or not.
- A `value-mapping` attribute with the possible values, such as `canonical`, `never-empty`, `never-null` can be used to filter values after matching is done. For instance, a parameter configured with `value-mapping="never-empty"` and matched with the empty string value will not put the empty string in the map.

Route precedence

The order of route declaration is important as it affects on how rules are matched. Sometimes, the same request could be matched by several routes and the routing table is ambiguous.

```
<route path="/foo">
  <route-param qname="gtn:handler">
    <value>portal</value>
  </route-param>
</route>
<route path="/{gtn:path}">
  <path-param encoding="preserve-path" qname="gtn:path">
    <pattern>.*</pattern>
  </path-param>
</route>
```

In that case, the request path "/foo" will always be matched by the first rule before the second rule. This can be misleading since the map (gtn:path=foo) would be rendered as "/foo" as well and would not be matched by the first rule. Such ambiguit can happen, it can be desirable or not.

Route nesting

Route nesting is possible and often desirable as it helps to:

- Factor common parameters in a common rule.
- Perform more efficient matching as the match of the common rule is done once for all the sub routes.

```
<route path="/foo">
  <route-param qname="gtn:handler">
    <value>portal</value>
  </route-param>
  <route path="/bar">
    <route-param qname="gtn:path">
      <value>bar</value>
    </route-param>
  </route>
  <route path="/juu">
    <route-param qname="gtn:path">
      <value>juu</value>
```

```
    </route-param>
  </route>
</route>
```

- The request path "/foo/bar" is mapped to the (gtn:handler=portal,gtn:path=bar) map.
- The request path "/foo/juu" is mapped to the (gtn:handler=portal,gtn:path=juu) map.
- The request path "/foo" is not mapped as non leaf routes do not perform matches.

Integrate to GateIn WebUI framework

Routing

GateIn defines a set of parameters in its routing table, for each client request, the mapping engine processes the request path and return the defined parameters with their values as a Map<QualifiedName, String>

gtn:handler

The gtn:handler names is one of the most important qualified name as it determines which handler will take care of the request processing just after the controller has determined the parameter map. The handler value is used to make a lookup in the handler map of the controller. An handler is a class that extends the `WebRequestHandler` class and implements the `execute(ControllerContext)` method. Several handlers are available by default:

- portal: process aggregated portal requests.
- upload/download: process file upload and download.
- standalone: process standalone portal requests.
- legacy: handle legacy URL redirection (see *the section called “Legacy handler”*).
- default: HTTP redirection to the default portal of the container.
- staticResource: serve static resources like image, CSS or JavaScript and more in **portal.war** (see *the section called “Static resource handler”*).

gtn:sitetype / gtn:sitename / gtn:path

Those qualified names drives a request for the portal handler. They are used to determine which site to show and which path to resolve against a navigation. For instance, the (gtn:sitetype=portal,gtn:sitename=classic,gtn:path=home) instruct the portal handler to show the home page of the classic portal site.

gtn:lang

This parameter shows which language used in the URL for the portal handler. This is a new feature offered, now language can be specified on URL. It means that users can bookmark that URL (with the information about language) or he can changed the language simply by modifying the URL address.

gtn:componentid / gtn:action / gtn:objectid

The webui parameters used by the portal handler for managing webui component URLs for portal applications (but not for portlet applications).

Rendering

The **controller** is designed to render a Map<QualifiedName, String> as an HTTP URL according to its routing table, but to integrate it for easy usage in WebUI Framework of GateIn, you need some more components:

PortalURL

`PortalURL` plays a similar role at the portal level. Its main role is to abstract the creation of an URL for a resource managed by the portal.

```
public abstract class PortalURL<R, U extends PortalURL<U>>
{
  ...
}
```

The `PortalURL` declaration may seem a bit strange at first sight with two generic types: `U` and `R`.

- The `R` generic type represents the type of the resource managed by the portal.
- The `U` generic type is also described as **self bound generic type**. This design pattern allows a class to return subtypes of itself in the class declaring the generic type. Java Enums are based on this principle (`class Enum<E extends Enum<E>>`).

A portal URL has various methods but certainly the most important method is the `toString()` method that generates an URL targeting to the resource. The remaining methods are `getter` and `setter` used to mutate the URL configuration, those options will affect the URL representation when it is generated.

- resource: the mandatory resource associated with the URL.
- locale: the optional locale used in the URL allowing the creation of bookmarkable URL containing a language.

- confirm: the optional confirmation message displayed by the portal in the context of the portal UI.
- ajax: the ajax option allowing an ajax invocation of the URL.

Obtaining a PortalURL

`PortalURL` objects are obtained from `RequestContext` instance, such as the `PortalRequestContext`, or the `PortletRequestContext`. Usually, those are obtained thanks to the `getCurrentInstance` method of the `RequestContext` class:

```
RequestContext ctx = RequestContext.getCurrentInstance();
```

`PortalURL` are created via to the `createURL` method that takes an input as a resource type. The resource type is usually a constant and type-safe object that allows to retrieve the `PortalURL` subclasses:

```
RequestContext ctx = RequestContext.getCurrentInstance();
PortalURL<R, U> url = ctx.createURL(type);
```

In reality, you will use a concrete type constant and have instead more concrete code like:

```
RequestContext ctx = RequestContext.getCurrentInstance();
NodeURL url = ctx.createURL(NodeURL.TYPE);
```

Note

The `NodeURL.TYPE` is actually declared as `new ResourceType<NavigationResource, NodeURL>()` that can be described as a `type-literal` object emulated by a Java anonymous inner class. Such literal was introduced by Neil Gafter as Super Type Token and popularized by Google Guice as Type Literal. It is an interesting way to create a literal representing a kind of Java type.

NodeURL

The `NodeURL` class is one of the subclass of `PortalURL` that is specialized in navigation node resources:

```
public class NodeURL extends PortalURL<NavigationResource, NodeURL>
```

```
{
  ...
}
```

The `NodeURL` class does not carry any generic types of its super class, which means that a `NodeURL` is type-safe and you do not have to worry about generic types.

Using a `NodeURL` is pretty straightforward:

```
NodeURL url = RequestContext.getCurrentInstance().createURL(NodeURL.TYPE);
url.setResource(new NavigationResource("portal", "classic, "home"));
String s = url.toString();
```

The `NodeURL` subclass contains the specialized `setter` methods to make its usage even easier:

```
UserNode node = ...;
NodeURL url = RequestContext.getCurrentInstance().createURL(NodeURL.TYPE);
url.setNode(node);
String s = url.toString();
```

ComponentURL

The `ComponentURL` subclass is another specialization of `PortalURL` that allows the creation of WebUI components URLs. `ComponentURL` is commonly used to trigger WebUI events from client side:

```
<% def componentURL = uicomponent.event(...); /*or uicomponent.url(...) */ %>
  <a href=$componentURL>Click me</a>
```

Normally, you should not have to deal with it as the WebUI framework has already an abstraction for managing URL known as `URLBuilder`. The `URLBuilder` implementation delegates URL creation to `ComponentURL` objects.

Portlet URLs

Portlet URLs API implementation delegates to the portal `ComponentURL` (via the portlet container SPI). It is possible to control the language in the URL from a `PortletURL` object by setting the `gtn:lang` property:

- When the property value is set to a value returned by the `Locale#toString()` method for locale objects having a non null language value and a null variant value, the URL generated by the `PortletURL#toString()` method will contain the locale in the URL.
- When the property value is set to an empty string, the generated URL will not contain a language. If the incoming URL was carrying a language, this language will be erased.
- When the property value is not set, it will not affect the generated URL.

```
PortletURL url = resp.createRenderURL();
url.setProperty("gtn:lang", "fr");
writer.print("<a href='" + url + "'>French</a>");
```

Webui `URLBuilder`

This internal API used to create URL works as usual and delegates to the `PortletURL` API when the framework is executed in a portlet, and delegates to a `ComponentURL` API when the framework is executed in the portal context. The API has been modified to take in account the language in URL with two properties on the builder:

- locale: a locale for setting on the URL.
- removeLocale: a boolean for removing the locale present on the URL.

Groovy Templates

In a Groovy template, the mechanism to create an URL is the same as the way of APIs above, however a splash of integration has been done to make creation of NodeURL simpler. A closure is bound under the `nodeurl` name and is available for invocation anytime. It will simply create a NodeURL object and return it:

```
UserNode node = ...;
NodeURL url = nodeurl();
url.setNode(node);
String s = url.toString();
```

The `nodeurl` closure is bound to Groovy template in `WebuiBindingContext`.

```
// Closure nodeurl()
```

```
put("nodeurl", new Closure(this)
{
  @Override
  public Object call(Object[] args)
  {
    return context.createURL(NodeURL.TYPE);
  }
});
```

Changes and migration from GateIn 3.1.x

The navication controller implies a migration of the client code that is coupled to several internal APIs of GateIn. The major impact is related to anything dealing with URL:

- Creation of an URL representing a resource managed by the portal: navigation node or UI component.
- Using HTTP request related information.

Migration of navigation node URL

Using free form node

The previous code for creating navigation node was like:

```
String uri = Util.getPortalRequestContext().getPortalURI() + "home";
```

The new code will look like:

```
PortalURL nodeURL = nodeurl();
NavigationResource resource = new NavigationResource(SiteType.PORTAL,
 pcontext.getPortalOwner(), "home");
String uri = nodeURL.setResource(resource).toString();
```

Using UserNode object

The previous code for creating navigation node was like:

```
UserNode node = ...;
```

```
String uri = Util.getPortalRequestContext().getPortalURI() + node.getURI()";
```

The new code will look like

```
UserNode node = ...;
PortalURL nodeURL = nodeurl();
String uri = nodeURL.setNode(node).toString();
```

Security changes

Security configuration needs to be changed to keep the flexibility added by the navigation controller. In particular, the authentication does not depend anymore on path specified in `web.xml` but relies on the security mandated by the underlying resource instead. Here are the noticeable changes for security:

- Authentication is now triggered on the "`/login`" URL when it does not have a username or a password specified. Therefore, the URL `/login?initialURI=/classic/home` is (more or less) equivalent to `/private/classic/home`.
- When a resource cannot be viewed due to security constraint.
  - If the user is not logged, the authentication will be triggered.
  - Otherwise, a special page (the usual one) will be displayed instead.

Default handler

Redirection to the default portal used to be done by the `index.jsp` JSP page. This is not the case anymore, the *index.jsp* file has been removed and the welcome file in `web.xml` was removed, too. Instead a specific handler in the routing table has been configured, the sole role of this handler is to redirect the request to the default portal when no other request has been matched previously:

```
<controller>
  ...
  <route path="/">
    <route-param qname="gtn:handler">
      <value>default</value>
    </route-param>
  </route>
</controller>
```

Legacy handler

Legacy URLs such as `/public/...` and `/private/...` are now emulated to determine the best resource with the same resolution algorithm, but instead of displaying the page, it will make an HTTP 302 redirection to the correct URL. This handler is present in the controller configuration. There is a noticeable difference between the two routes.

- The public redirection attempts to find a node with the legacy resolution algorithm without authentication, which means that secured nodes will not be resolved and the redirection of a secured node will likely redirect to another page. For instance, resolving the URL */public/classic/administration/registry* path will likely resolve to another node if the user is not authenticated and is not in the platform administrator group.
- The private redirection performs first an authentication before doing the redirection. In that case, the */private/classic/administration/registry* path will be redirected to the */portal/groups/:platform:administrators/administration/registry* page if the user has the sufficient security rights.

Static resource handler

The "/" mapping for the "default" servlet is now replaced by mapping for the `org.exoplatform.portal.application.PortalController` servlet. It means that you need a handler (*org.exoplatform.portal.application.StaticResourceRequestHandler*) to serve static resources like image, CSS or JavaScript files in `portal.war`. And it should be configured, and extended easily thanks to the `controller.xml` file. This file can be overridden and can be changed and reloaded at runtime (WebAppController is MBean with some operations, such as `reloadConfiguration()`).

Declare StaticResourceHandler in `controller.xml`

```
<route path="/{gtn:path}">
  <route-param qname="gtn:handler">
    <value>staticResource</value>
  </route-param>
  <path-param encoding="preserve-path" qname="gtn:path">
    <pattern>.*\.(jpg|png|gif|ico|css)</pattern>
  </path-param>
</route>
```

And you do not need these kinds of the following mapping in the *web.xml* in `portal.war` anymore.

```
<servlet-mapping>
  <servlet-name>default</servlet-name>
```

```
  <url-pattern>*.jpg</url-pattern>
</servlet-mapping>
...
```

portal.war's web.xml changes

DoLoginServlet declaration:

```
<servlet>
  <servlet-name>DoLoginServlet</servlet-name>
  <servlet-class>org.exoplatform.web.login.DoLoginServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>DoLoginServlet</servlet-name>
  <url-pattern>/dologin</url-pattern>
</servlet-mapping>
```

Delare **portal servlet** as the default servlet:

```
<servlet-mapping>
   <servlet-name>portal</servlet-name>
   <url-pattern>/</url-pattern>
</servlet-mapping>
```

Some mapping declarations for portal servlet are unused,so you should remove them: **/private/* /public/* /admin/* /upload/* /download/***

Add some security constraints:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>user authentication</web-resource-name>
      <url-pattern>/dologin</url-pattern>
      <url-pattern>/standalone/*</url-pattern>
      <url-pattern>/groups/*</url-pattern>
      <url-pattern>/users/*</url-pattern>
...
  </web-resource-collection>
```

```
</security-constraint>
```

You can remove the *index.jsp* file, and its declaration in the *web.xml* file thank to the default request handler:

```
<welcome-file-list>
  <welcome-file>/index.jsp</welcome-file>
</welcome-file-list>
```

Dashboard changes

There are several important changes to take in account:

- Dashboard are now bound to a single URL (/users/root by default) and dashboard pages are leaf of this path.
- Dashboard lifecycle can be decoupled (create or destroy) from the identity creation in a configurable manner in `UserPortalConfigService` and exposed in `configuration.properties` under `gatein.portal.idm.createuserportal` and `gatein.portal.idm.destroyuserportal`.
- By default, dashboard are not created when a user is registered.
- A dashboard is created when the user accesses his dashboard URL.

Remove unused files

portal-unavailable.jsp: this file is presented before if a user goes to a non-available portal. But now the server sends a 404 status code instead.

portal-warning.jsp: this file is not used in any places.

Portlet development

Portlet Primer

JSR-168 and JSR-286 overview

The Java Community Process (JCP) uses Java Specification Requests (JSRs) to define proposed specifications and technologies designed for the Java platform.

The Portlet Specifications aim at defining portlets that can be used by any *JSR-168 (Portlet 1.0)* [http://www.jcp.org/en/jsr/detail?id=168] or *JSR-286 (Portlet 2.0)* [http://www.jcp.org/en/jsr/detail?id=286] portlet container.

Most Java EE (Enterprise Edition) portals include at least one compliant portlet container, and GateIn 3.2 is no exception. In fact, GateIn 3.2 includes a container that supports both versions.

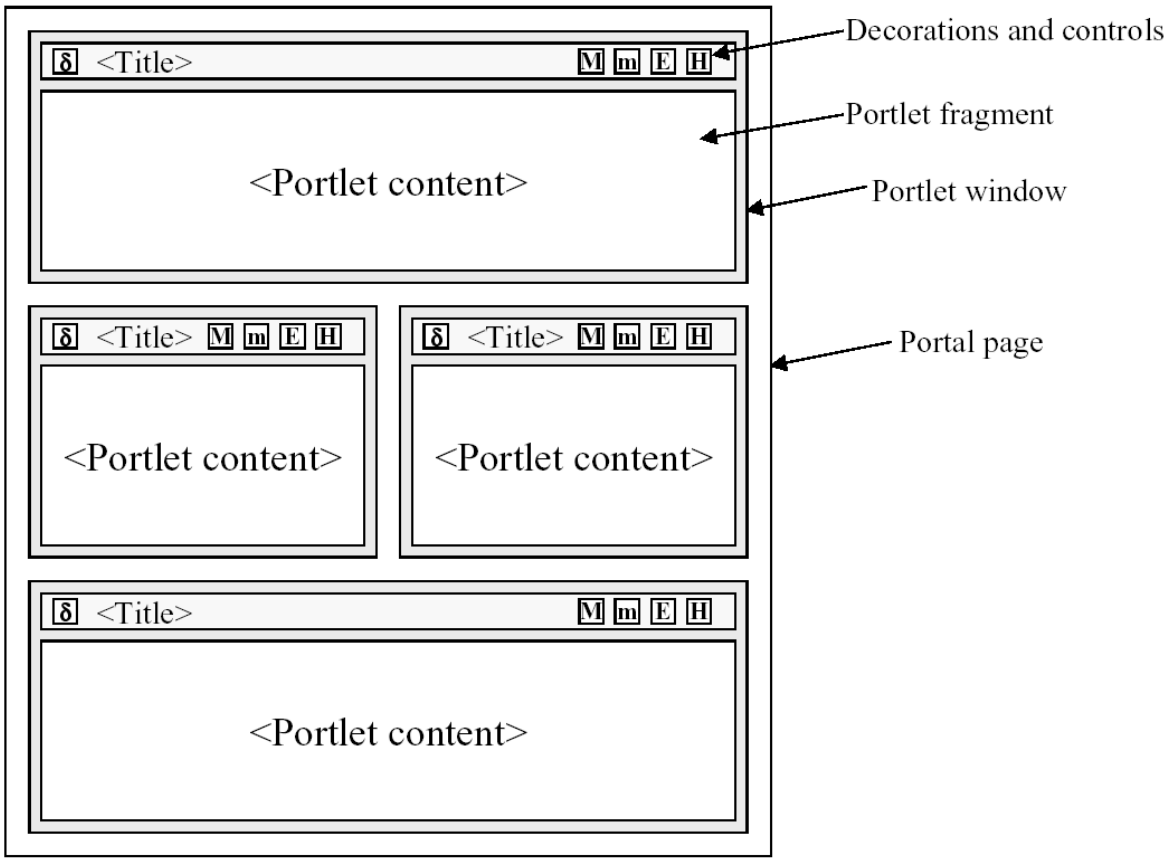
This chapter gives a brief overview of the Portlet Specifications, but portlet developers are strongly encouraged to read the *JSR-286 Portlet Specification* [http://www.jcp.org/en/jsr/detail?id=286] .

GateIn 3.2 is fully JSR-286 compliant. Any JSR-168 or JSR-286 portlet operates as it is mandated by the respective specifications inside the portal.

Portal Pages

A portal can be considered as a series of web pages with different *areas* within them. Those areas contain different *windows* and each *window* contains portlet:

The diagram below visually represents this nesting:



Rendering Modes

A portlet can have different view modes. Three modes are defined by the JSR-286 specification:

View
: Generate the markup reflecting the current state of the portlet.

Edit
: Allow you to customize the behavior of the portlet.

Help
: Provide information to the user as to how to use the portlet.

Window States

Window states are an indicator of how much page space a portlet consumes on any given page. The three states defined by the JSR-168 specification are:

Normal
: A portlet shares this page with other portlets.

Minimized
: A portlet may show very little information, or none at all.

Maximized
: A portlet may be the only portlet displayed on this page.

Tutorials

The tutorials contained in this chapter are targeted towards portlet developers. It is also recommended that developers read and understand the *JSR-286 Portlet Specification* [http://www.jcp.org/en/jsr/detail?id=286] .

Maven

This example is using Maven to compile and build the web archive. Maven versions can be downloaded from *maven.apache.org* [http://maven.apache.org/download.html]

Deploying your first Portlet

This section describes how to deploy a portlet in GateIn 3.2. A sample portlet called `SimplestHelloWorld` is located in the `examples` directory at the root of your GateIn 3.2 binary package. This sample is used in the following examples.

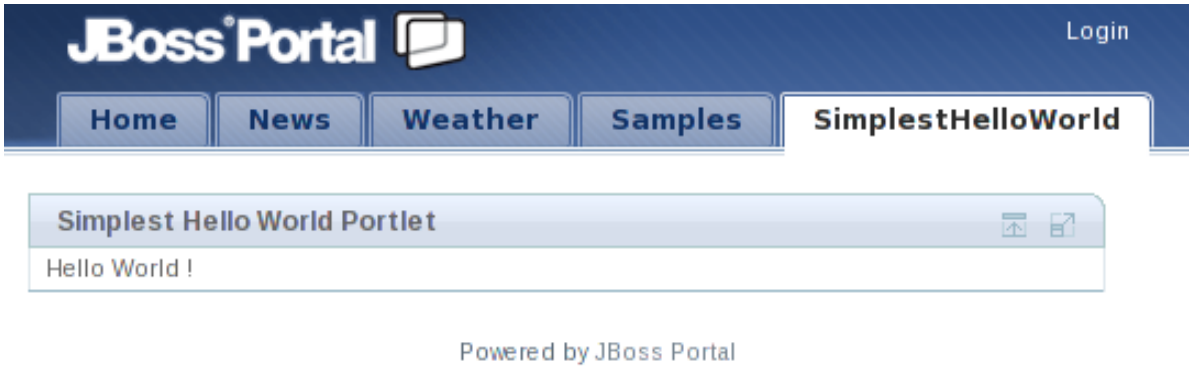
Compiling

To compile and package the application:

1. Navigate to the `SimplestHelloWorld` directory and execute:

```
mvn package
```

2. If the compile is successfully packaged, the result will be available in: `SimplestHelloWorld/target/SimplestHelloWorld-0.0.1.war` .
3. Copy the package file into `JBOSS_HOME/server/default/deploy`.
4. Start the JBoss Application Server (if it is not already running).
5. Create a new portal page and add the portlet to it.



Package Structure

Like other Java EE applications, GateIn 3.2 portlets are packaged in the .war files. A typical portlet .war file can include servlets, resource bundles, images, HTML, JavaServer Pages (JSP), and other static or dynamic files.

The following is an example of the directory structure of the `SimplestHelloWorld` portlet:

```
|-- SimplestHelloWorld-0.0.1.war
|   `-- WEB-INF
|       |-- classes
|       |   `-- org
|       |       `-- gatein
|       |           `-- portal
|       |               `-- examples
|       |                   `-- portlets
|       |                       `-- SimplestHelloWorldPortlet.class
|       |-- portlet.xml
|       `-- web.xml
```

1. The compiled Java class implementing *javax.portlet.Portlet* (through *javax.portlet.GenericPortlet*)
2. This is the mandatory descriptor files for portlets. It is used during deployment.
3. This is the mandatory descriptor for web applications.

Portlet Class

Below is the `SimplestHelloWorldPortlet/src/main/java/org/gatein/portal/examples/portlets/SimplestHelloWorldPortlet.java` Java source:

```
package org.gatein.portal.examples.portlets;

import java.io.IOException;
import java.io.PrintWriter;

import javax.portlet.GenericPortlet;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;

public class SimplestHelloWorldPortlet extends GenericPortlet
{
   public void doView(RenderRequest request,
```

```
                    RenderResponse response) throws IOException
  {
    PrintWriter writer = response.getWriter();
    writer.write("Hello World !");
    writer.close();
  }
}
```

1. All portlets must implement the `javax.portlet.Portlet` interface. The portlet API provides a convenient implementation of this interface.

   The `javax.portlet.Portlet` interface uses the `javax.portlet.GenericPortlet` class which implements the `Portlet render` method to dispatch to abstract mode-specific methods. This makes it easier to support the standard portlet modes.

   `Portlet render` also provides a default implementation for the `processAction`, `init` and `destroy` methods. It is recommended to extend `GenericPortlet` for most cases.
2. If only the `view` mode is required, only the `doView` method needs to be implemented. The `GenericPortletrender` implementation calls our implementation when the `view` mode is requested.
3. Use the *RenderResponse* to obtain a writer to be used to produce content.
4. Write the markup to display.
5. Close the writer.

Markup Fragments

Portlets are responsible for generating markup fragments, as they are included on a page and are surrounded by other portlets. This means that a portlet outputing HTML must not output any markup that cannot be found in a `<body>` element.

Application Descriptors

GateIn 3.2 requires certain descriptors to be included in a portlet WAR file. These descriptors are defined by the Jave EE (`web.xml`) and Portlet Specification (`portlet.xml`).

Below is an example of the `SimplestHelloWorldPortlet/WEB-INF/portlet.xml` file. This file must adhere to its definition in the JSR-286 Portlet Specification. More than one portlet application may be defined in this file:

```
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
```

```
                                    http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  version="2.0">
  <portlet>
    <portlet-name>SimplestHelloWorldPortlet</portlet-name>
    <portlet-class>
      org.gatein.portal.examples.portlets.SimplestHelloWorldPortlet
    </portlet-class>
    <supports>
      <mime-type>text/html</mime-type>
    </supports>
    <portlet-info>
       <title>Simplest Hello World Portlet</title>
    </portlet-info>
  </portlet>
</portlet-app>
```

1. Define the portlet name. It does not have to be the class name.
2. The Fully Qualified Name (`FQN`) of your portlet class must be declared here.
3. The `<supports>` element declares all of the markup types that a portlet supports in the `render` method. This is accomplished via the `<mime-type>` element, which is required for every portlet.

   The declared MIME types must match the capability of the portlet. It allows administrators to pair which modes and window states are supported for each markup type.

   This does not have to be declared as all portlets must support the `view` portlet mode.

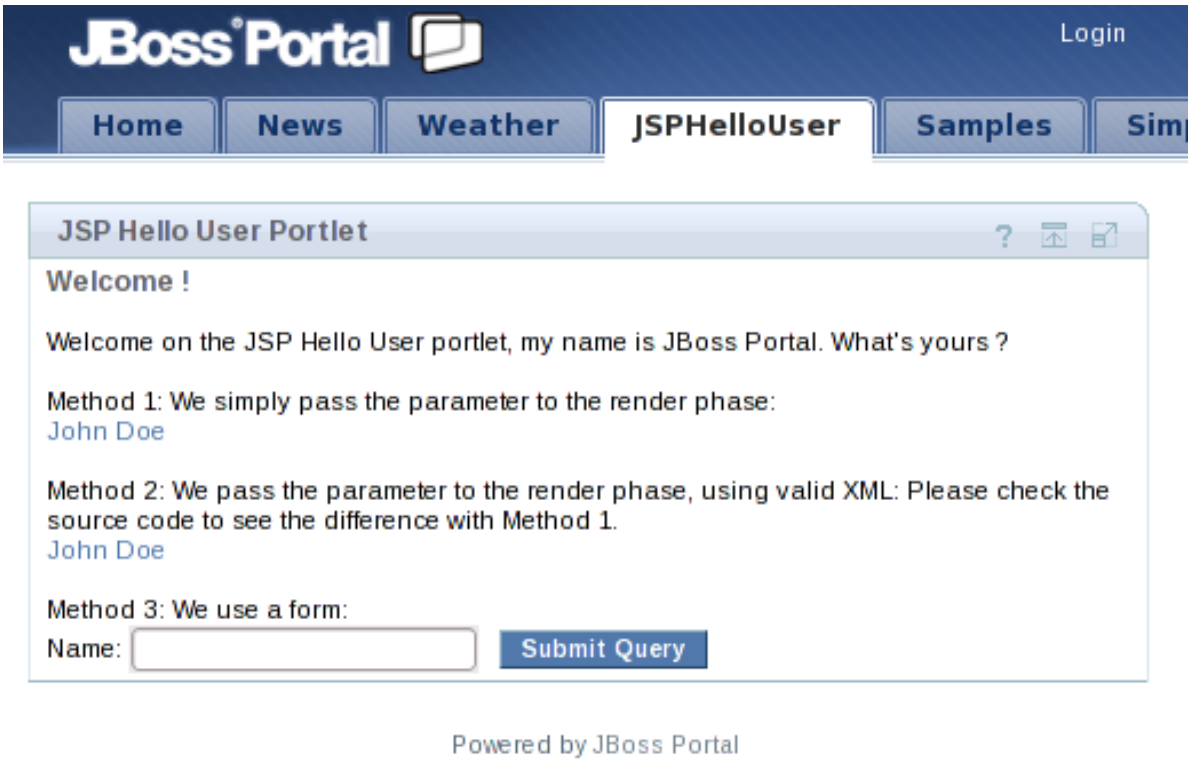
   Use the `<mime-type>` element to define which markup type the portlet supports. In the example above, this is `text/html`. This section tells the portal to only output HTML.
4. When rendered, the portlet's title is displayed as the header in the portlet window, unless it is overridden programmatically. In the example above, the title would be `Simplest Hello World Portlet` .

JavaServer Pages Portlet Example

This section discusses:

1. Add more features to the previous example.
2. Use a JSP page to render the markup.
3. Use the portlet tag library to generate links to the portlet in different ways.
4. Use the other standard portlet modes.

1. The example used in this section can be found in the `JSPHelloUser` directory.
2. Execute **mvn package** in this directory.
3. Copy `JSPHelloUser/target/JSPHelloUser-0.0.1.war` to the `deploy` directory of JBoss Application Server.
4. Select the new `JSPHelloUser` tab in your portal.



Note

The `EDIT` button only appears with logged-in users, which is not the case in the screenshot.

Package Structure

The package structure in this tutorial does not much differ from the previous example, with the exception of adding some JSP files detailed later.

The JSPHelloUser portlet contains the mandatory portlet application descriptors. The following is an example of the directory structure of the JSPHelloUser portlet:

```
JSPHelloUser-0.0.1.war
   |-- META-INF
   |   |-- MANIFEST.MF
   |-- WEB-INF
   |   |-- classes
```

```
|   |   `-- org
|   |       `-- gatein
|   |           `-- portal
|   |               `-- examples
|   |                   `-- portlets
|   |                       `-- JSPHelloUserPortlet.class
|   |-- portlet.xml
|   `-- web.xml
`-- jsp
    |-- edit.jsp
    |-- hello.jsp
    |-- help.jsp
    `-- welcome.jsp
```

Portlet Class

The code below is from the `JSPHelloUser/src/main/java/org/gatein/portal/examples/portlets/JSPHelloUserPortlet.java` Java source. It is split in different pieces.

```
package org.gatein.portal.examples.portlets;

import java.io.IOException;

import javax.portlet.ActionRequest;
import javax.portlet.ActionResponse;
import javax.portlet.GenericPortlet;
import javax.portlet.PortletException;
import javax.portlet.PortletRequestDispatcher;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
import javax.portlet.UnavailableException;

public class JSPHelloUserPortlet extends GenericPortlet
{

  public void doView(RenderRequest request, RenderResponse response)
      throws PortletException, IOException
  {
    String sYourName = (String) request.getParameter("yourname");
    if (sYourName != null)
    {
      request.setAttribute("yourname", sYourName);
      PortletRequestDispatcher prd =
```

```
          getPortletContext().getRequestDispatcher("/jsp/hello.jsp");
        prd.include(request, response);
    }
    else
    {
                PortletRequestDispatcher prd = getPortletContext().getRequestDispatcher("/jsp/
welcome.jsp");
        prd.include(request, response);
    }
  }
...
```

1. Override the *doView* method (as in the first tutorial).
2. This entry attempts to obtain the value of the render parameter named `yourname`. If defined, it should redirect to the `hello.jsp` JSP page or to the `welcome.jsp` JSP page.
3. Get a request dispatcher on a file located within the web archive.
4. Perform the inclusion of the markup obtained from the JSP.

Like the `VIEW` portlet mode, the specification defines two other modes; `EDIT` and `HELP`.

These modes need to be defined in the `portlet.xml` descriptor. This enables the corresponding buttons on the portlet's window.

The generic portlet that is inherited dispatches different views to the methods: `doView` , `doHelp` and `doEdit`.

```
...
     protected void doHelp(RenderRequest rRequest, RenderResponse rResponse) throws
 PortletException, IOException,
       UnavailableException
  {
    rResponse.setContentType("text/html");
    PortletRequestDispatcher prd = getPortletContext().getRequestDispatcher("/jsp/help.jsp");
    prd.include(rRequest, rResponse);
  }

     protected void doEdit(RenderRequest rRequest, RenderResponse rResponse) throws
 PortletException, IOException,
       UnavailableException
  {
    rResponse.setContentType("text/html");
    PortletRequestDispatcher prd = getPortletContext().getRequestDispatcher("/jsp/edit.jsp");
```

```
    prd.include(rRequest, rResponse);
  }
...
```

Portlet calls happen in one or two phases: when the portlet is rendered and when the portlet is actioned *and then* rendered.

An action phase is a phase where containing some state changes. The render phase will have access to render parameters that will be passed each time the portlet is refreshed (with the exception of caching capabilities).

The code to be executed during an action has to be implemented in the *processAction* method of the portlet.

```
...
        public void processAction(ActionRequest aRequest, ActionResponse aResponse) throws
 PortletException, IOException,
       UnavailableException
  {
    String sYourname = (String) aRequest.getParameter("yourname");
    aResponse.setRenderParameter("yourname", sYourname);
  }
...
```

1. `processAction` is the method from `GernericPorlet` to override for the *action* phase.
2. Here the parameter is retrieved through an *action URL* .
3. The value of `yourname` is kept to make it available in the rendering phase. The previous line simply copies action parameters to a render parameter for this example.

JSP files and the Portlet Tag Library

The `help.jsp` and `edit.jsp` files are very simple. Note that CSS styles are used as defined in the portlet specification. This ensures that the portlet will render successfully within the theme and across portal vendors.

```
<div class="portlet-section-header">Help mode</div>
<div class="portlet-section-body">This is the help mode where you can find useful information.</
div>
```

```
<div class="portlet-section-header">Edit mode</div>
```

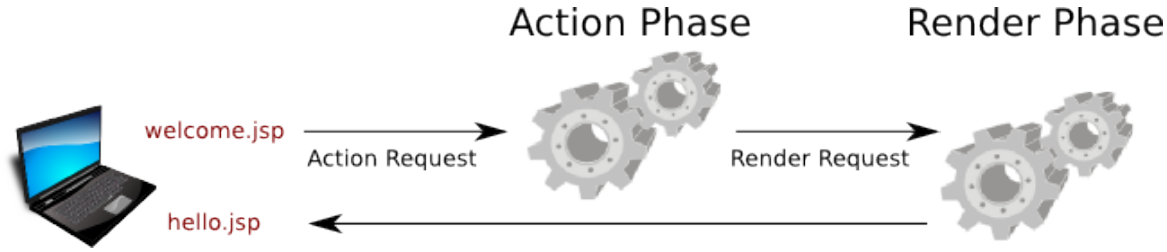
```
<div class="portlet-section-body">This is the edit mode where you can change your portlet preferences.</div>
```

The landing page contains the links and form to call the portlet:

```
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>

<div class="portlet-section-header">Welcome !</div>

<br/>

<div class="portlet-font">Welcome on the JSP Hello User portlet,
my name is GateIn Portal. What's yours ?</div>

<br/>

<div class="portlet-font">Method 1: We simply pass the parameter to the render phase:<br/>
<a href="<portlet:renderURL><portlet:param name="yourname" value="John Doe"/>
            </portlet:renderURL>">John Doe</a></div>

<br/>

<div class="portlet-font">Method 2: We pass the parameter to the render phase, using valid XML:
Please check the source code to see the difference with Method 1.
<portlet:renderURL var="myRenderURL">
   <portlet:param name="yourname" value='John Doe'/>
</portlet:renderURL>
<br/>
<a href="<%= myRenderURL %>">John Doe</a></div>

<br/>

<div class="portlet-font">Method 3: We use a form:<br/>

<portlet:actionURL var="myActionURL"/>
<form action="<%= myActionURL %>" method="POST">
        <span class="portlet-form-field-label">Name:</span>
        <input class="portlet-form-input-field" type="text" name="yourname"/>
        <input class="portlet-form-button" type="Submit"/>
</form>
</div>
```

1. The portlet taglib which needs to be declared.
2. The first method shown here is the simplest one. `portlet:renderURL` will create a URL that calls the render phase of the current portlet and append the result at the place of the markup (within a tag). A parameter is also added directly to the URL.
3. In this method, the `var` attribute is used. This avoids having one XML tag within another. Instead of printing the url, the `portlet:renderURL` tag will store the result in the referenced variable (`myRenderURL`).
4. The variable `myRenderURL` is used like any other JSP variable.
5. The third method mixes the form submission and action request. Again, a temporary variable is used to put the created URL into.
6. The action URL is used in the HTML form.

In the third method, the action phase is triggered first, then the render phase is triggered, which outputs some content back to the web browser based on the available render parameters.



JSF example using the JBoss Portlet Bridge

To write a portlet using JSF, it is required to have a 'bridge'. This software allows developers to write a portlet application as if it was a JSF application. The bridge then negotiates the interactions between the two layers.

An example of the JBoss Portlet Bridge is available in `examples/JSFHelloUser`. The configuration is slightly different from a JSP application. This example can be used as a base to configure instead of creating a new application.

As in any JSF application, the file `faces-config.xml` is required. It must contain the following information:

```
<faces-config>
...
   <application>
     <view-handler>org.jboss.portletbridge.application.PortletViewHandler</view-handler>
     <state-manager>org.jboss.portletbridge.application.PortletStateManager</state-manager>
  </application>
...
</faces-config>
```

The portlet bridge libraries must be available and are usually bundled with the `WEB-INF/lib` directory of the web archive.

The other differences as compared to a regular portlet application can be found in the portlet descriptor. All details about it can be found in the JSR-301 specification that the JBoss Portlet Bridge implements.

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
                      http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
  version="2.0">
  <portlet>
    <portlet-name>JSFHelloUserPortlet</portlet-name>
    <portlet-class>javax.portlet.faces.GenericFacesPortlet</portlet-class>
    <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>view</portlet-mode>
      <portlet-mode>edit</portlet-mode>
      <portlet-mode>help</portlet-mode>
    </supports>
    <portlet-info>
      <title>JSF Hello User Portlet</title>
    </portlet-info>

    <init-param>
      <name>javax.portlet.faces.defaultViewId.view</name>
      <value>/jsf/welcome.jsp</value>
    </init-param>

    <init-param>
      <name>javax.portlet.faces.defaultViewId.edit</name>
      <value>/jsf/edit.jsp</value>
    </init-param>

    <init-param>
      <name>javax.portlet.faces.defaultViewId.help</name>
      <value>/jsf/help.jsp</value>
    </init-param>

  </portlet>
</portlet-app>
```

1. All JSF portlets define `javax.portlet.faces.GenericFacesPortlet` as the portlet class. This class is part of the JBoss Portlet Bridge.
2. This is a mandatory parameter to define what's the default page to display.
3. This parameter defines which page to display on the 'edit' mode.
4. This parameter defines which page to display on the 'help' mode.

Global porlet.xml file

Global portlet.xml usecase

The Portlet Specification introduces PortletFilter as a standard approach to extend the behaviors of portlet objects. For example, a filter can transform the content of portlet requests and portlet responses. According to the Portlet Specification, normally there are three steps in setting up a portlet filter:

1. Implement a PortletFilter object
2. Define the filter in portlet application deployment descriptor
3. Define the filter mapping in portlet definitions

Two first steps are quite simple and easy to be done, however, at the step 3, developers/ administrators need to replicate the filter mapping in many portlet definitions, that makes work error and tedious in several use cases. The global portlet feature is designed to compensate such limitation.

Global metadata

The Global metadata is declared in the *portlet.xml* file conforming with Portlet 2.0 's XSD.

```
<portlet-app version="1.0" xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd http://
java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd">

</portlet-app>
```

Location

The path to the global *portlet.xml* is value of *gatein.portlet.config* in the *configuration.properties* file and varied by hosting application servers.

For Tomcat: *TOMCAT_HOME/gatein/conf/portlet.xml*

For JBoss: *JBOSS_HOME/server/default/conf/gatein/portlet.xml*

Global metadata elements

The global *portlet.xml* file conforms to the schema of the portlet deployment descriptor defined in the Portlet Specification with some restrictions. In this file, the following elements are supported:

1. Portlet Filter
2. Portlet Mode
3. Window State

Portlet filter

Portlet filter mappings declared in the global *portlet.xml* file are applied across portlet applications. With the XML configuration below, the filter ApplicationMonitoringFilter involves in request handling on any deployed portlet.

```
<filter>
    <filter-name>org.exoplatform.portal.application.ApplicationMonitoringFilter</filter-name>
   <filter-class>org.exoplatform.portal.application.ApplicationMonitoringFilter</filter-class>
   <lifecycle>ACTION_PHASE</lifecycle>
   <lifecycle>RENDER_PHASE</lifecycle>
   <lifecycle>EVENT_PHASE</lifecycle>
   <lifecycle>RESOURCE_PHASE</lifecycle>
</filter>
```

Application Monitoring Filter supports four lifecycle phases as the order below: *ACTION_PHASE/ EVENT_PHASE/ RENDER_PHASE/ RESOURCE_PHASE* and records statistic information on deployed portlets. The filter alternates actual monitoring mechanism in WebUI Framework.

Portlet Mode and Window State

The global *portlet.xml* file is considered as an alternative place to declare custom Portlet Modes and Window States.

Gadget development

Gadgets

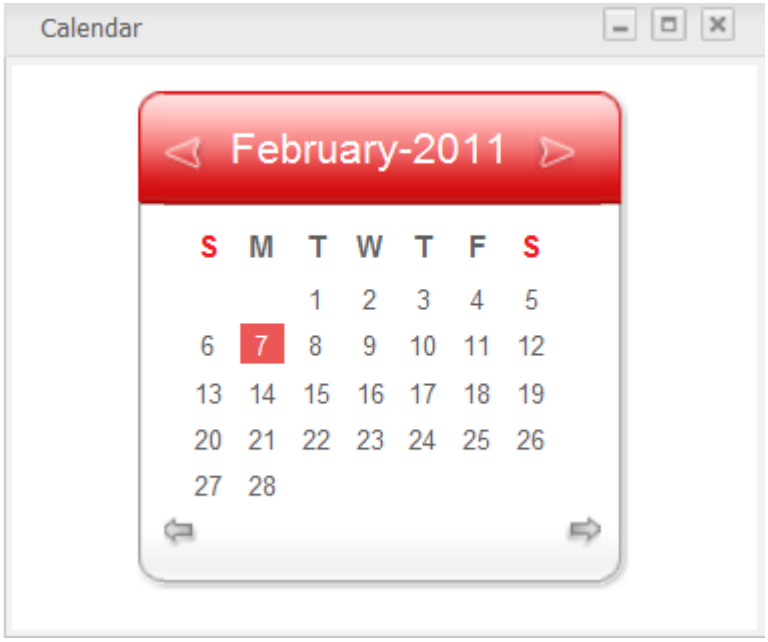
A gadget is a mini web application, embedded in a web page and running on an application server platform. These small applications help users perform various tasks.

GateIn 3.2 supports gadgets, such as Todo, Calendar, Calculator, Weather Forecasts and RSS Reader.

Default Gadgets:

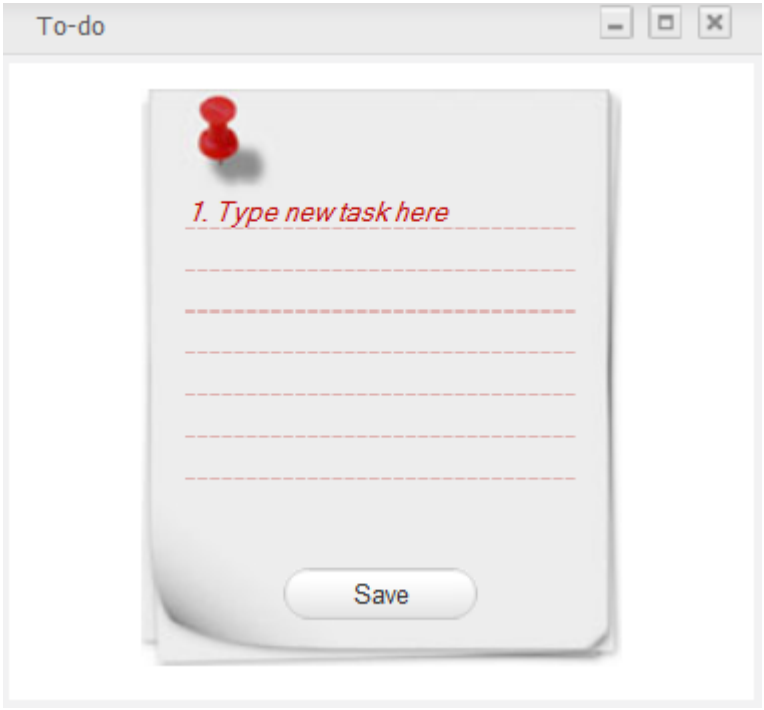
Calendar

The calendar gadget allows you to switch easily between daily, monthly and yearly views. Also, this gadget is customizable to match your portal's theme.



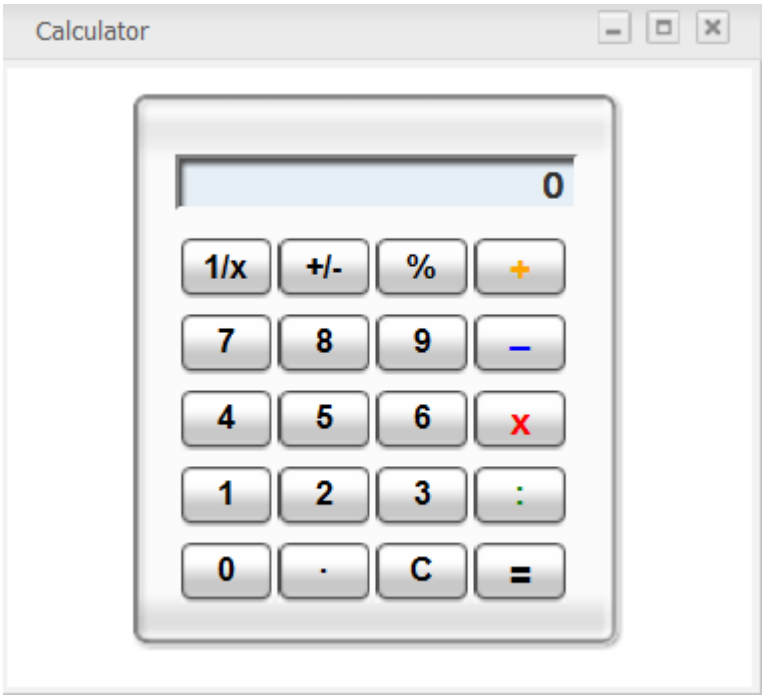
ToDo

This application helps you organize your day and work group. It is designed to keep track of your tasks in a convenient and transparent way. Tasks can be highlighted with different colors.



Calculator

This mini-application lets you perform the most basic arithmetic operations and can be themed to match the rest of your portal.



RSS Reader

An RSS reader, or aggregator collects content from various, user-specified feed sources and displays them in one location. This content can include, but is not limited to, news headlines,

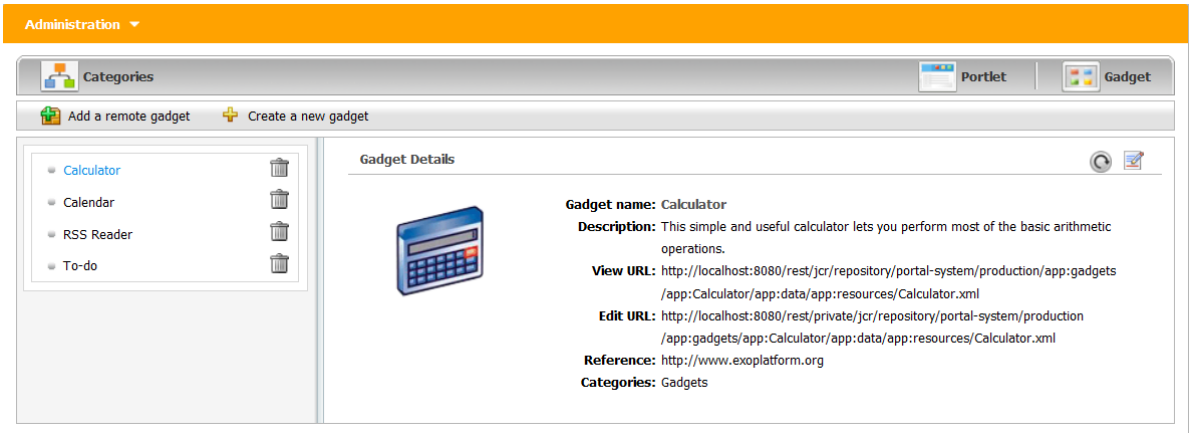
blog posts or email. The RSS Reader gadget displays this content in a single window on your Portal page.

More Gadgets

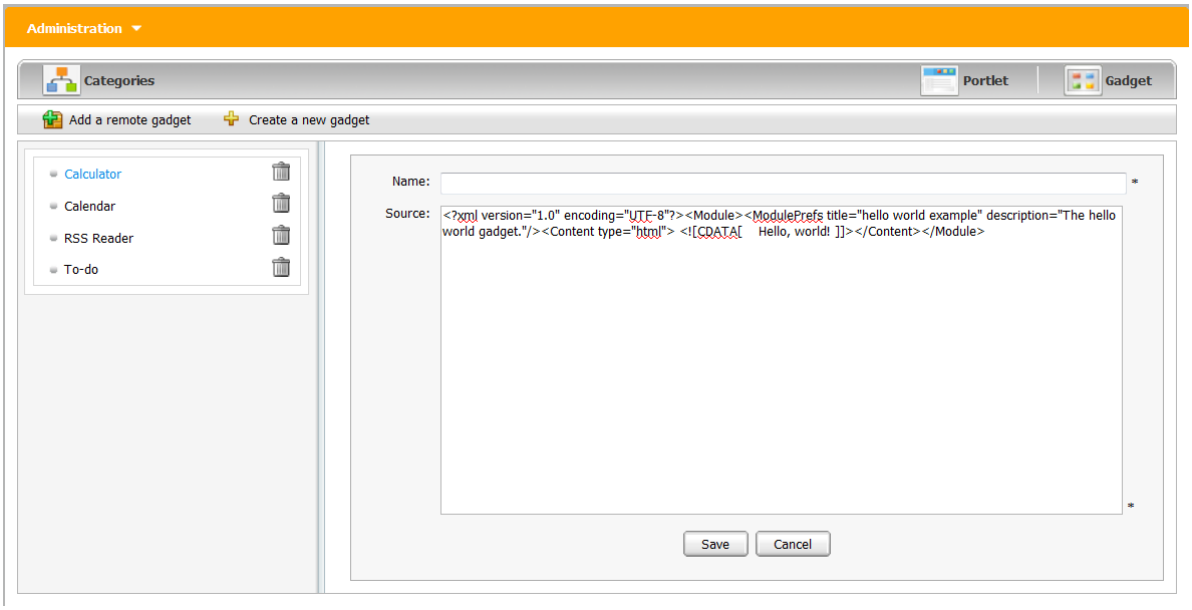
Further gadgets can be obtained from the *Google Gadget* [http://www.google.com/ig/directory?synd=open] site. GateIn 3.2 is compatible with most of the gadgets available here.

The following sections require more textual information.

Existing Gadgets

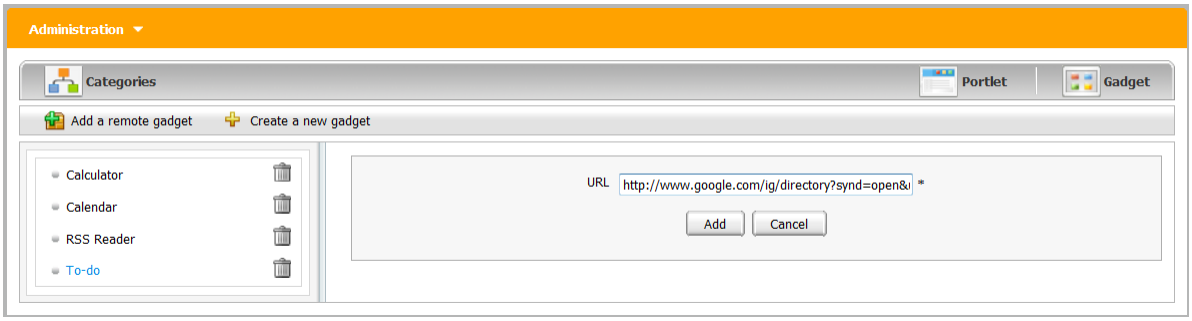


Create a new Gadget



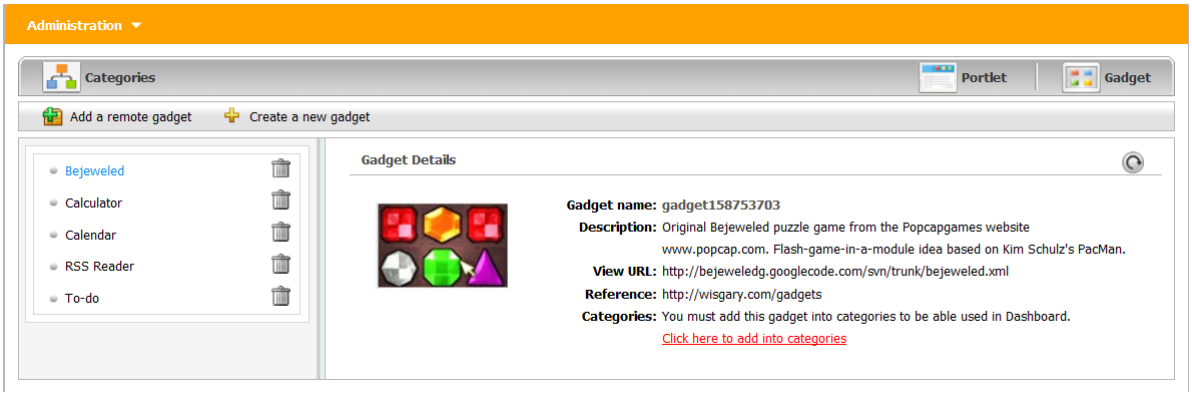
Remote Gadget

This is the reference to a remote gadget (stock one).



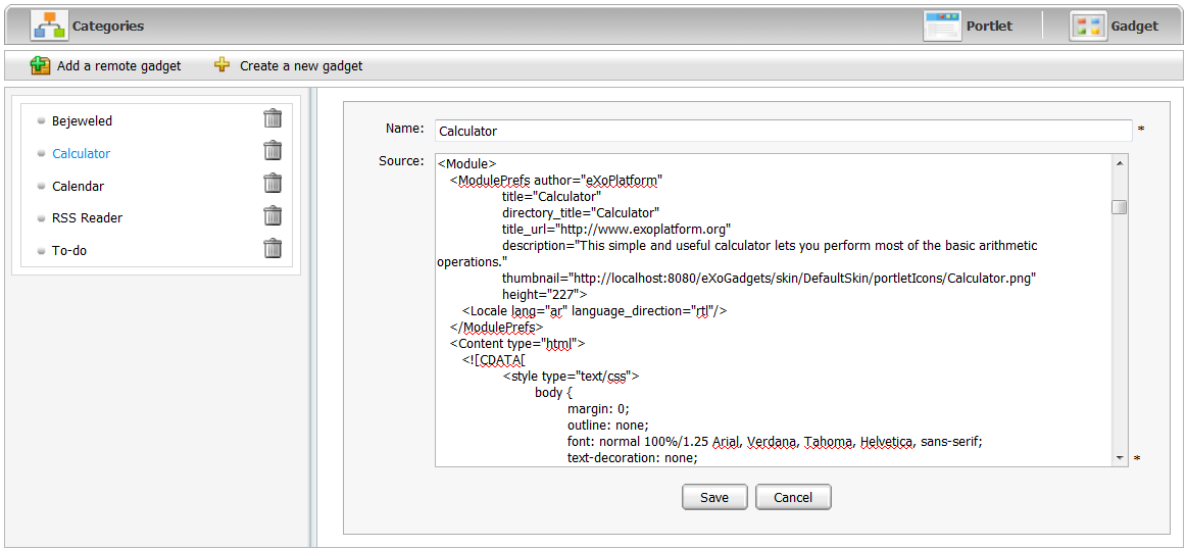
Gadget Importing

After referencing the gadget successfully, import it into the local repository.



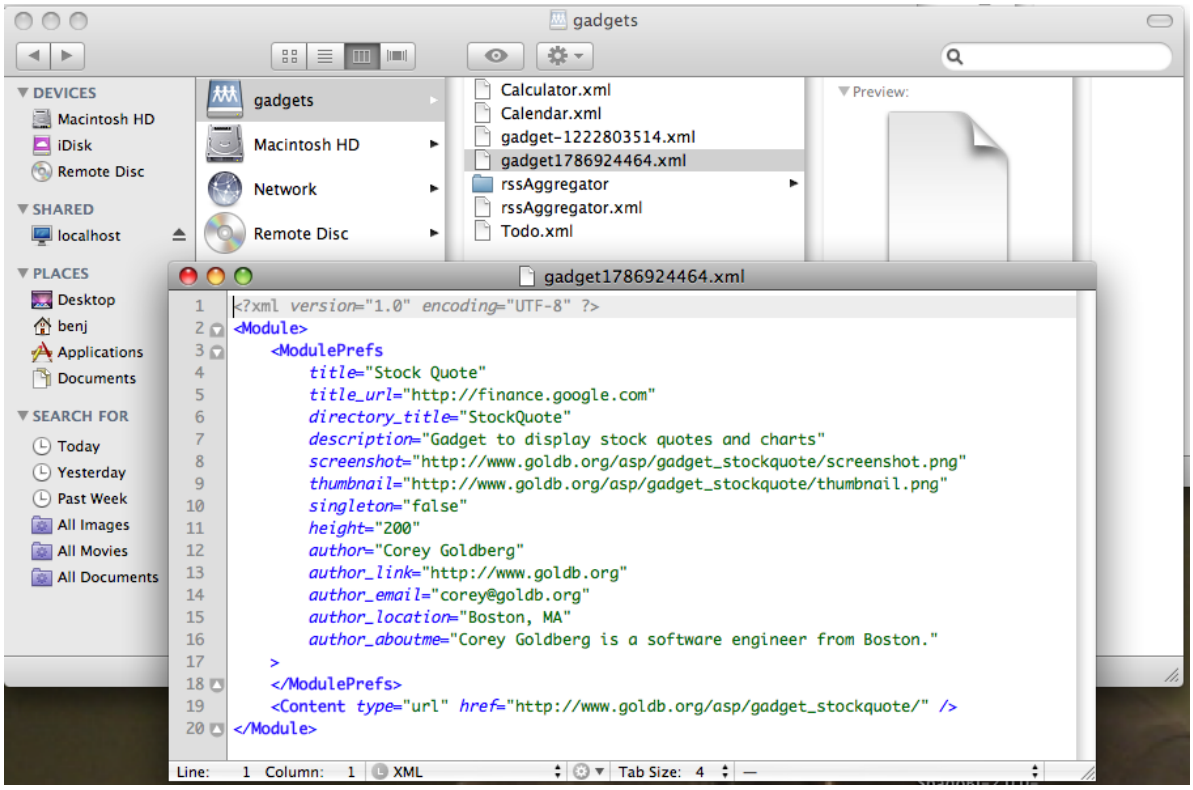
Gadget Web Editing

Modify it from the Web where the Gadget was imported:



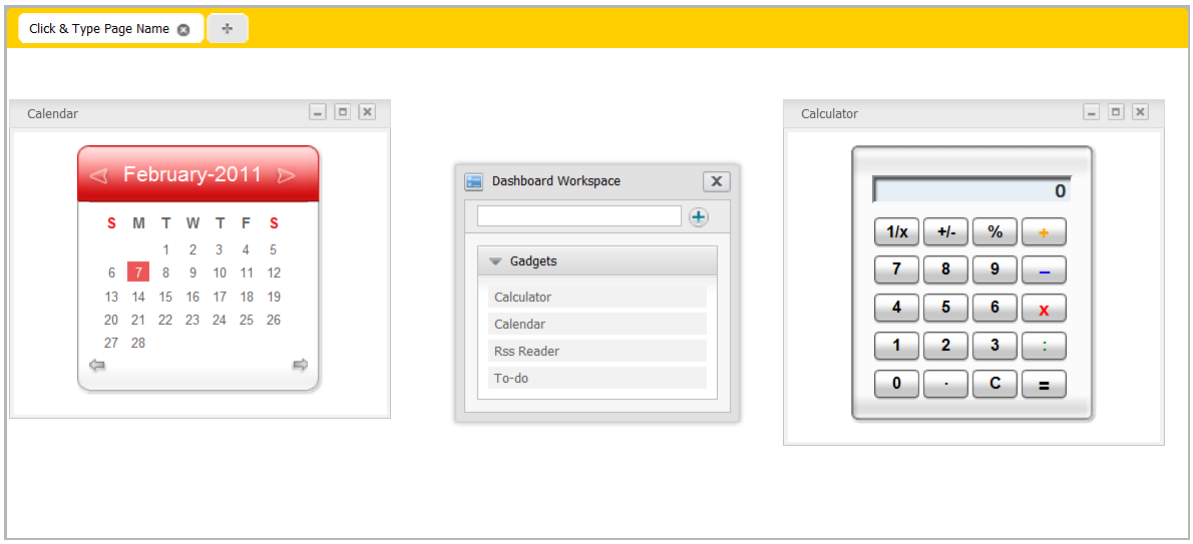
Gadget IDE Editing

Edit it from your IDE thanks to the WebDAV protocol:



Dashboard Viewing

View it from the Dashboard when you drag and drop the Gadget from listing to the dashboard.



Set up a Gadget Server

Virtual servers for gadget rendering

GateIn 3.2 recommends using two virtual hosts for security. If the gadget is running on a different domain other than the container (the website that 'contains' the app), it is unable to interfere with the portal by modifying the code or cookies.

An example would hosting the portal from **http://www.sample.com** and the gadgets from **http://www.samplemodules.com**.

To do this, configure a parameter called *gadgets.hostName*. The value is the *path/to/gadgetServer* in `GadgetRegisteryService`:

```
<component>
  <key>org.exoplatform.application.gadget.GadgetRegistryService</key>
  <type>org.exoplatform.application.gadget.jcr.GadgetRegistryServiceImpl</type>
  <init-params>
    <value-param>
      <name>gadgets.hostName</name>
      <description>Gadget server url</description>
      <value>http://localhost:8080/GateInGadgetServer/gadgets/</value>
    </value-param>
  </init-params>
</component>
```

It is also possible to have multiple rendering servers. This helps to balance the rendering load across multiple servers.

When deploying on the same server, ensure the gadget initiates before anything that calls it (for example; the webapp `GateInGadgets` which uses `org.exoplatform.application.gadget.GadgetRegister`).

Configuration

Security key

In GateIn, the gadget container is using three security files for authentication and authorization gadgets:

- *key.txt*
- *oauthkey.pem*
- *oauthkey_pub.pem*

By default, they are located in the *tomcat/gatein/conf/gadgets* folder and are configured by system variables in the *tomcat/gatein/conf/configuration.properties* file:

```
gatein.gadgets.securitytokenkeyfile=${gatein.conf.dir}/gadgets/key.txt
gatein.gadgets.signingkeyfile=${gatein.conf.dir}/gadgets/oauthkey.pem
```

In case you have other files, you can change these variables to point to them.

The *key.txt* file contains a secret key used to encrypt the security token used for the user authentication. When starting GateIn, this file is read via the *gatein.gadgets.securitytokenkeyfile* path. In case the *key.txt* file is not found, GateIn automatically generates a new *key.txt* one and save it to the *gatein.gadgets.securitytokenkeyfile* path.

oauthkey.pem and *oauthkey_pub.pem* are a key pair of RSA cryptography standard. *oauthkey.pem* is known as a private key and *oauthkey_pub.pem* is a public key. They are the default keys of the gadget container which OAuth gadgets will use to authorize with external service providers.

Gadget proxy and concat configuration

These servers have to be on the same domain as the gadget server. You can configure the container in `eXoGadgetServer:/WEB-INF/classes/containers/default/container.js`.

```
"gadgets.content-rewrite" : {
  "include-urls": ".*",
  "exclude-urls": "",
  "include-tags": ["link", "script", "embed", "img", "style"],
  "expires": "86400",
  "proxy-url": "http://localhost:8080/eXoGadgetServer/gadgets/proxy?url=",
  "concat-url": "http://localhost:8080/eXoGadgetServer/gadgets/concat?"
},
```

Proxy

To allow external gadgets when the server is behind a proxy, add the following code to the beginning of the JVM:

```
-Dhttp.proxyHost=proxyhostURL -Dhttp.proxyPort=proxyPortNumber -Dhttp.proxyUser=someUserName -Dhttp.proxyPassword=somePassword
```

Authentication and Identity

Predefined User Configuration

Overview

To specify the initial Organization configuration, the content of `02portal.war:/WEB-INF/conf/organization/organization-configuration.xml` should be edited. This file uses the portal XML configuration schema. It lists several configuration plugins.

Plugin for adding users, groups and membership types

The plugin of type `org.exoplatform.services.organization.OrganizationDatabaseInitializer` is used to specify the list of membership types/groups/users to be created.

The **checkDatabaseAlgorithm** initialization parameter determines how the database update is performed.

If its value is set to **entry**, it means that each user, group and membership listed in the configuration is checked each time GateIn 3.2 is started. If the entry does not exist in the database yet, it is created. If the **checkDatabaseAlgorithm** parameter value is set to **empty**, the configuration data will be updated to the database only if the database is empty.

Membership types

The predefined membership types are specified in the **membershipType** field of the **OrganizationConfig** plugin parameter.

Note

See `02portal.war:/WEB-INF/conf/organization/organization-configuration.xml` for the full content.

```
<field name="membershipType">
 <collection type="java.util.ArrayList">
  <value>
   <object type="org.exoplatform.services.organization.OrganizationConfig$MembershipType">
    <field name="type">
     <string>member</string>
    </field>
    <field name="description">
     <string>member membership type</string>
    </field>
```

```
      </object>
    </value>
    <value>
     <object type="org.exoplatform.services.organization.OrganizationConfig$MembershipType">
       <field name="type">
         <string>owner</string>
       </field>
       <field name="description">
         <string>owner membership type</string>
       </field>
     </object>
    </value>
    <value>
     <object type="org.exoplatform.services.organization.OrganizationConfig$MembershipType">
       <field name="type">
         <string>validator</string>
       </field>
       <field name="description">
         <string>validator membership type</string>
       </field>
      </object>
    </value>
  </collection>
</field>
```

Groups

The predefined groups are specified in the **group** field of the **OrganizationConfig** plugin parameter.

```
<field name="group">
  <collection type="java.util.ArrayList">
    <value>
      <object type="org.exoplatform.services.organization.OrganizationConfig$Group">
       <field name="name">
         <string>portal</string>
       </field>
       <field name="parentId">
         <string></string>
       </field>
       <field name="type">
         <string>hierachy</string>
       </field>
```

```
      <field name="description">
        <string>the /portal group</string>
      </field>
    </object>
  </value>
  <value>
    <object type="org.exoplatform.services.organization.OrganizationConfig$Group">
      <field name="name">
        <string>community</string>
      </field>
      <field name="parentId">
        <string>/portal</string>
      </field>
      <field name="type">
        <string>hierachy</string>
      </field>
      <field name="description">
        <string>the /portal/community group</string>
      </field>
    </object>
  </value>
  ...
 </collection>
</field>
```

Users

The predefined users are specified in the **membershipType** field of the **OrganizationConfig** plugin parameter.

```
<field name="user">
 <collection type="java.util.ArrayList">
  <value>
    <object type="org.exoplatform.services.organization.OrganizationConfig$User">
      <field name="userName"><string>root</string></field>
      <field name="password"><string>exo</string></field>
      <field name="firstName"><string>root</string></field>
      <field name="lastName"><string>root</string></field>
      <field name="email"><string>exoadmin@localhost</string></field>
    <field name="groups"><string>member:/admin,member:/user,owner:/portal/admin</string></
field>
    </object>
  </value>
```

```
    <value>
      <object type="org.exoplatform.services.organization.OrganizationConfig$User">
        <field name="userName"><string>exo</string></field>
        <field name="password"><string>exo</string></field>
        <field name="firstName"><string>site</string></field>
        <field name="lastName"><string>site</string></field>
        <field name="email"><string>exo@localhost</string></field>
        <field name="groups"><string>member:/user</string></field>
      </object>
    </value>
    ...
  </collection>
</field>
```

Plugin for monitoring user creation

The plugin of type `org.exoplatform.services.organization.impl.NewUserEventListener` specifies which groups all the newly created users should become members of. It specifies the groups and the memberships to use (while the group is just a set of users, a membership type represents a user's role within a group). It also specifies a list of users that should not be processed (i.e. administrative users like 'root').

Note

The terms 'membership' and 'membership type' refer to the same thing, and are used interchangeably.

```
<component-plugin>
  <name>new.user.event.listener</name>
  <set-method>addListenerPlugin</set-method>
  <type>org.exoplatform.services.organization.impl.NewUserEventListener</type>
  <description>this listener assign group and membership to a new created user</description>
  <init-params>
    <object-param>
      <name>configuration</name>
      <description>description</description>
      <object type="org.exoplatform.services.organization.impl.NewUserConfig">
        <field  name="group">
          <collection type="java.util.ArrayList">
            <value>
              <object type="org.exoplatform.services.organization.impl.NewUserConfig$JoinGroup">
                <field name="groupId"><string>/user</string></field>
                <field name="membership"><string>member</string></field>
              </object>
```

```
          </value>
        </collection>
      </field>
      <field  name="ignoredUser">
        <collection type="java.util.HashSet">
          <value><string>exo</string></value>
          <value><string>root</string></value>
          <value><string>company</string></value>
          <value><string>community</string></value>
        </collection>
      </field>
    </object>
  </object-param>
 </init-params>
</component-plugin>
```

Authentication Token Configuration

What is Token Service?

Token Service is used in authentication.

The token system prevents user account information being sent in the clear text mode within inbound requests. This increases authentication security.

The token service allows administrators to create, delete, retrieve and clean tokens as required. The service also defines a validity period of any given token. The token becomes invalid once this period expires.

Implementing the Token Service API

All token services used in the GateIn 3.2 authentication must be implemented by subclassing an **AbstractTokenService** abstract class. The following **AbstractTokenService** methods represent the contract between authentication runtime, and a token service implementation.

```
 public Token getToken(String id) throws PathNotFoundException, RepositoryException;
 public Token deleteToken(String id) throws PathNotFoundException, RepositoryException;
 public String[] getAllTokens();
 public long getNumberTokens() throws Exception;
            public String createToken(Credentials credentials) throws
IllegalArgumentException,NullPointerException;
      public Credentials validateToken(String tokenKey, boolean remove) throws
NullPointerException;
```

Configuring token services

The token services configuration includes specifying the token validity period. The token service is configured as a portal component (in the portal scope, as opposed to the root scope - more about that in Foundations chapter).

In the example below, *CookieTokenService* is a subclass of **AbstractTokenService**, so it has a property which specifies the validity period of the token.

The token service will initialize this validity property by looking for an `init-param` named **service.configuration**.

This property must have three values.

```
<component>
  <key>org.exoplatform.web.security.security.CookieTokenService</key>
  <type>org.exoplatform.web.security.security.CookieTokenService</type>
  <init-params>
   <values-param>
      <name>service.configuration</name>
      <value>jcr-token</value>
      <value>7</value>
      <value>DAY</value>
    </values-param>
  </init-params>
</component>
```

1. Service name
2. Amount of time
3. Unit of time

In this case, the service name is **jcr-token** and the token expiration time is one week.

GateIn 3.2 supports *four* time units:

1. `SECOND`
2. `MINUTE`
3. `HOUR`
4. `DAY`

PicketLink IDM integration

GateIn 3.2 uses the PicketLink IDM component to keep the necessary identity information, such as users, groups, memberships. While legacy interfaces are still used (org.exoplatform.services.organization) for identity management, there is a wrapper implementation that delegates to the PicketLink IDM framework.

This section does not provide information about PicketLink IDM and its configuration. Please refer to the appropriate project documentation (*http://jboss.org/picketlink/IDM.html*) for further information.

Note

It is important to fully understand the concepts behind this framework design before changing the default configuration.

The identity model represented in '**org.exoplatform.services.organization**' interfaces and the one used in **PicketLink IDM** have some major differences.

The org.exoplatform.services.organization interface stores and manages information of users, groups or memberships, user profiles, relationships and retrieval. The management of org.exoplatform.services.organization interface is divided into many layers, such as model object, data access object and authentication.

For example: **PicketLink IDM** provides greater abstraction. It is possible for groups in **IDM** framework to form memberships with many parents (which requires recursive ID translation), while the GateIn model allows only pure tree-like membership structures.

Additionally, the GateIn *membership* concept needs to be translated into the IDM *Role* concept. Therefore, the **PicketLink IDM** model is used in a limited way. All these translations are applied by the integration layer.

Configuration files

The main configuration file is **idm-configuration.xml**:

```
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd  http://
www.exoplaform.org/xml/ns/kernel_1_0.xsd"
          xmlns="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd">

  <component>
     <key>org.exoplatform.services.organization.idm.PicketLinkIDMService</key>
    <type>org.exoplatform.services.organization.idm.PicketLinkIDMServiceImpl</type>
    <init-params>
      <value-param>
        <name>config</name>
```

```
          <value>war:/conf/organization/idm-config.xml</value>
        </value-param>
        <value-param>
          <name>portalRealm</name>
          <value>realm${container.name.suffix}</value>
        </value-param>
      </init-params>
  </component>


  <component>
    <key>org.exoplatform.services.organization.OrganizationService</key>
      <type>org.exoplatform.services.organization.idm.PicketLinkIDMOrganizationServiceImpl</
type>
    <init-params>
    <object-param>
      <name>configuration</name>
      <object type="org.exoplatform.services.organization.idm.Config">
        <field name="useParentIdAsGroupType">
          <boolean>true</boolean>
        </field>

        <field name="forceMembershipOfMappedTypes">
          <boolean>true</boolean>
        </field>

        <field name="pathSeparator">
          <string>.</string>
        </field>

        <field name="rootGroupName">
          <string>GTN_ROOT_GROUP</string>
        </field>

        <field name="groupTypeMappings">
          <map type="java.util.HashMap">
            <entry>
              <key><string>/</string></key>
              <value><string>root_type</string></value>
            </entry>

            <!-- Sample mapping -->
            <!--
            <entry>
```

```
            <key><string>/platform/*</string></key>
            <value><string>platform_type</string></value>
          </entry>
          <entry>
            <key><string>/organization/*</string></key>
            <value><string>organization_type</string></value>
          </entry>
          -->

        </map>
      </field>

      <field name="associationMembershipType">
        <string>member</string>
      </field>

      <field name="ignoreMappedMembershipType">
        <boolean>false</boolean>
      </field>
    </object>
  </object-param>
 </init-params>


 </component>

</configuration>
```

1 The **org.exoplatform.services.organization.idm.PicketLinkIDMServiceImpl** service has the following options:

config
: (value-param)

  PicketLink IDM configuration file

hibernate.properties
: (properties-param)

  A list of hibernate properties used to create SessionFactory that will be injected to JBoss Identity IDM configuration registry.

hibernate.annotations
: A list of annotated classes that will be added to Hibernate configuration.

hibernate.mappings
A list of .xml files that will be added to the hibernate configuration as mapping files.

jndiName
(value-param)

If the 'config' parameter is not provided, this parameter will be used to perform the JNDI lookup for IdentitySessionFactory.

portalRealm
(value-param)

The realm name that should be used to obtain the proper IdentitySession. The default is 'PortalRealm'.

2 The **org.exoplatform.services.organization.idm.PicketLinkIDMOrganizationServiceImpl** key is a main entrypoint implementing **org.exoplatform.services.organization.OrganizationService** and is dependant on **org.exoplatform.services.organization.idm.PicketLinkIDMService**

org.exoplatform.services.organization.idm.PicketLinkIDMOrganizationServiceImpl service has the following options defined as fields of object-param of type **org.exoplatform.services.organization.idm.Config**:

defaultGroupType
The name of the PicketLink IDM GroupType that will be used to store groups. The default is 'GTN_GROUP_TYPE'.

rootGroupName
The name of the PicketLink IDM Group that will be used as a root parent. The default is 'GTN_ROOT_GROUP'

passwordAsAttribute
This parameter specifies if a password should be stored using the PicketLink IDM Credential object or as a plain attribute. The default value is set to false.

useParentIdAsGroupType
This parameter stores the parent ID path as a group type in PicketLink IDM for any IDs not mapped with a specific type in 'groupTypeMappings'. If this option is set to false, and no mappings are provided under 'groupTypeMappings', only one group with the given name can exist in the GateIn 3.2 group tree.

pathSeparator
When 'userParentIdAsGroupType' is set to true, this value will be used to replace all "/" characters in IDs. The "/" character is not allowed in the group type name in PicketLink IDM.

associationMembershipType
: If this option is used, each Membership created with MembrshipType that is equal to the value specified here, will be stored in PicketLink IDM as the simple Group-User association.

groupTypeMappings
: This parameter maps groups added with GateIn 3.2 API as children of a given group ID, and stores them with a given group type name in PicketLink IDM.

  If the parent ID ends with "/*", all child groups will have the mapped group type. Otherwises, only direct (first level) children will use this type.

  This can be leveraged by LDAP if the LDAP DN is configured in PicketLink IDM to only store a specific group type. This will then store the given branch in the GateIn 3.2 group tree, while all other groups will remain in the database.

forceMembershipOfMappedTypes
: Groups stored in PicketLink IDM with a type mapped in 'groupTypeMappings' will automatically be members under the mapped parent. The Group relationships linked by the PicketLink IDM group association will not be necessary.

  This parameter can be set to false if all groups are added via GateIn 3.2 APIs. This may be useful with the LDAP configuration when being set to true, it will make every entry added to LDAP appear in GateIn 3.2. This, however, is not true for entries added via GateIn 3.2 management UI.

ignoreMappedMembershipType
: If "associationMembershipType" option is used, and this option is set to true, Membership with MembershipType configured to be stored as PicketLink IDM association will not be stored as PicketLink IDM Role.

Additionally, **JBossIDMOrganizationServiceImpl** uses those defaults to perform identity management operations.

- GateIn 3.2 User interface properties fields are persisted in JBoss Identity IDM using those attributes names: firstName, lastName, email, createdDate, lastLoginTime, organizationId, password (if password is configured to be stored as attribute).
- GateIn 3.2 Group interface properties fields are persisted in JBoss Identity IDM using those attributes names: label, description.
- GateIn 3.2 MembershipType interface properties fields are persisted in JBoss Identity IDM using those RoleType properties: description, owner, create_date, modified_date.

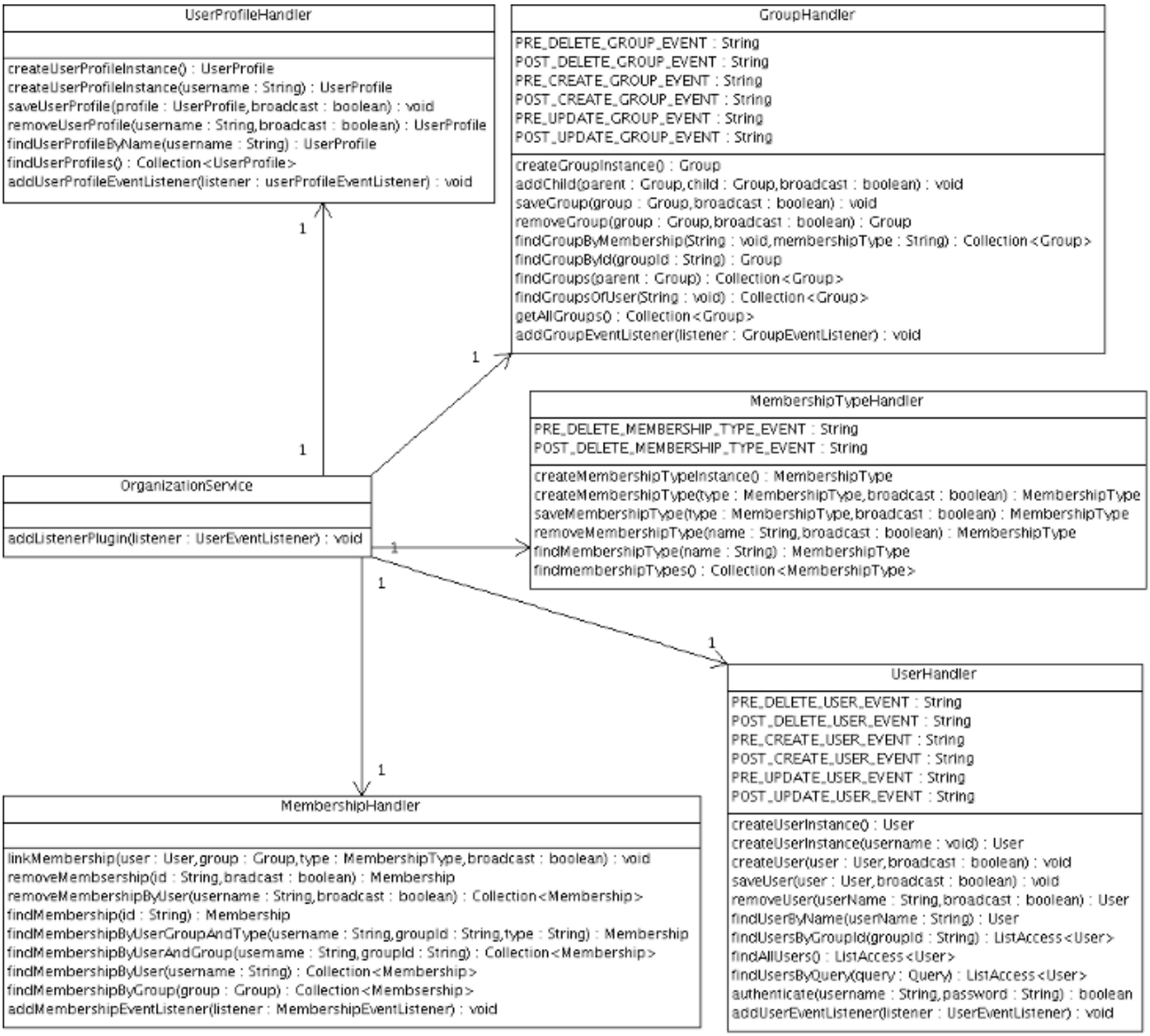
A sample **PicketLink IDM** configuration file is shown below. To understand all the options it contains, please refer to the PicketLink IDM Reference Guide.

```
<jboss-identity xmlns="urn:jboss:identity:idm:config:v1_0_beta"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="urn:jboss:identity:idm:config:v1_0_alpha identity-config.xsd">
   <realms>
      <realm>
         <id>PortalRealm</id>
         <repository-id-ref>PortalRepository</repository-id-ref>
         <identity-type-mappings>
            <user-mapping>USER</user-mapping>
         </identity-type-mappings>
      </realm>
   </realms>
   <repositories>
      <repository>
         <id>PortalRepository</id>
         <class>org.jboss.identity.idm.impl.repository.WrapperIdentityStoreRepository</class>
         <external-config/>
         <default-identity-store-id>HibernateStore</default-identity-store-id>
         <default-attribute-store-id>HibernateStore</default-attribute-store-id>
      </repository>
   </repositories>
   <stores>
      <attribute-stores/>
      <identity-stores>
         <identity-store>
            <id>HibernateStore</id>
            <class>org.jboss.identity.idm.impl.store.hibernate.HibernateIdentityStoreImpl</class>
            <external-config/>
            <supported-relationship-types>
               <relationship-type>JBOSS_IDENTITY_MEMBERSHIP</relationship-type>
               <relationship-type>JBOSS_IDENTITY_ROLE</relationship-type>
            </supported-relationship-types>
            <supported-identity-object-types>
               <identity-object-type>
                  <name>USER</name>
                  <relationships/>
                  <credentials>
                     <credential-type>PASSWORD</credential-type>
                  </credentials>
                  <attributes/>
                  <options/>
               </identity-object-type>
            </supported-identity-object-types>
```

```
            <options>
               <option>
                  <name>hibernateSessionFactoryRegistryName</name>
                  <value>hibernateSessionFactory</value>
               </option>
               <option>
                  <name>allowNotDefinedIdentityObjectTypes</name>
                  <value>true</value>
               </option>
               <option>
                  <name>populateRelationshipTypes</name>
                  <value>true</value>
               </option>
               <option>
                  <name>populateIdentityObjectTypes</name>
                  <value>true</value>
               </option>
               <option>
                  <name>allowNotDefinedAttributes</name>
                  <value>true</value>
               </option>
               <option>
                  <name>isRealmAware</name>
                  <value>true</value>
               </option>
            </options>
         </identity-store>
      </identity-stores>
   </stores>
</jboss-identity>
```

Organization API

The `exo.platform.services.organization` package has five main components: user, user profile, group, membership type and membership. There is an additional component that serves as an entry point into Organization API - `OrganizationService` component that provides the handling functionality for the five components.



The `User` component contains basic information about the user, such as username, password, first name, last name, and email. The `User Profile` component contains extra information about a user, such as user's personal information, and business information. You can also add additional information about a user if your application requires it. The `Group` component contains a group graph. The `Membership Type` component contains a list of predefined membership types. Finally, the `Membership` component connects a User, a Group and a Membership Type.

A user can have one or more memberships within a group, for example: the user A can have two memberships: 'member' and 'admin' in the group /user. A user belongs to a group if he has at least one membership in that group.

Exposing the Organization API to developers who have access to the handler objects provided by the OrganizationService component. These handler objects are used to manage each of the five components, including UserHandler, UserProfileHandler, GroupHandler, MembershipTypeHandler, and MembershipHandler.

The five central API components are really designed like persistent entities, and handlers are really specified like data access objects (DAO).

Organization API simply describes a contract, meaning it is not a concrete implementation. The described components are interfaces, allowing different concrete implementations. Practically, it means that you can replace the existing implementation with a different one.

Accessing User Profile

The following code retrieves the details for a logged-in user:

```
// Alternative context: WebuiRequestContext context =
 WebuiRequestContext.getCurrentInstance() ;
PortalRequestContext context = PortalRequestContext.getCurrentInstance() ;
// Get the id of the user logged
String userId = context.getRemoteUser();
// Request the information from OrganizationService:
OrganizationService orgService = getApplicationComponent(OrganizationService.class) ;
if (userId != null)
  {
  User user = orgService.getUserHandler().findUserByName(userId) ;
  if (user != null)
  {
    String firstName = user.getFirstName();
    String lastName = user.getLastName();
    String email = user.getEmail();
  }
}
```

Below are two alternatives for retrieving the Organization Service:

```
OrganizationService service = (OrganizationService)

 ExoContainerContext.getCurrentContainer().getComponentInstanceOfType(OrganizationService.class);
```

```
OrganizationService service = (OrganizationService)
   PortalContainer.getInstance().getComponentInstanceOfType(OrganizationService.class);
```

Single-Sign-On (SSO)

Overview

GateIn 3.2 provides some forms of Single-Sign-On (SSO) as an integration and aggregation platform.

When logging into the portal, users gain access to many systems through portlets using a single identity. In many cases, the portal infrastructure must be integrated with other SSO enabled systems. There are many different Identity Management solutions available. In most cases, each SSO framework provides a unique way to plug into a Java EE application.

Prerequisites

In this tutorial, the SSO server is installed in a Tomcat installation. Tomcat can be obtained from *http://tomcat.apache.org*.

All the packages required for setup can be found in a zip file located at: *https://repository.jboss.org/nexus/content/groups/public/org/gatein/sso/sso-packaging/1.0.1-GA/sso-packaging-1.0.1-GA.zip*. In this document, $GATEIN_SSO_HOME is called as the directory where the file is extracted.

It is recommended that you should not run any portal extensions that could override the data when manipulating the `gatein.ear` file directly.

Remove `$JBOSS_HOME/server/default/deploy/gatein-sample-extension.ear` and `$JBOSS_HOME/server/default/deploy/gatein-sample-portal.ear` which are packaged with GateIn 3.2 by default.

Central Authentication Service (CAS)

This Single-Sign-On plugin enables the seamless integration between GateIn 3.2 and the CAS Single-Sign-On Framework. Details about CAS can be found *here* [http://www.ja-sig.org/products/cas/].

The integration consists of two steps: installing/configuring a CAS server and setting up the portal to use the CAS server.

CAS server

First, set up the server to authenticate against the portal login module. In this example, the CAS server is installed on Tomcat.

Obtaining CAS

CAS can be downloaded from *http://www.jasig.org/cas/download*.

Extract the downloaded file into a suitable location. This location will be referred to as `$CAS_HOME` in the following example.

Modifying the CAS server

To configure the web archive as desired, the simplest way is to make the necessary changes directly in the CAS codebase.

Note

To complete these instructions, and perform the final build step, you will need the Apache Maven 2. You can get it *here* [http://maven.apache.org/download.html].

First, change the default authentication handler with the one provided by GateIn 3.2.

The CAS Server Plugin makes secure authentication callbacks to a RESTful service installed on the remote GateIn 3.2 server to authenticate a user.

In order for the plugin to function correctly, it needs to be properly configured to connect to this service. This configuration is done via the `cas.war/WEB-INF/deployerConfigContext.xml` file.

1. Open `CAS_HOME/cas-server-webapp/src/main/webapp/WEB-INF/deployerConfigContext.xml`
2. Replace:

```
<!--
  | Whereas CredentialsToPrincipalResolvers identify who it is some Credentials might authenticate,
    | AuthenticationHandlers actually authenticate credentials.  Here e declare the AuthenticationHandlers that
 | authenticate the Principals that the CredentialsToPrincipalResolvers identified.  CAS will try these handlers in turn
   | until it finds one that both supports the Credentials presented and succeeds in authenticating.
 +-->
<property name="authenticationHandlers">
  <list>
    <!--
     | This is the authentication handler that authenticates services by means of callback via SSL, thereby validating
     | a server side SSL certificate.
     +-->
                                                                <bean class="org.jasig.cas.authentication.handler.support.HttpBasedServiceCredentialsAuthenticationHandler"
          p:httpClient-ref="httpClient" />
    <!--
       | This is the authentication handler declaration that every CAS deployer will need to change before deploying CAS
```

```
        | into production.  The default SimpleTestUsernamePasswordAuthenticationHandler
authenticates UsernamePasswordCredentials
        | where the username equals the password.  You will need to replace this with an
AuthenticationHandler that implements your
     | local authentication strategy.  You might accomplish this by coding a new such handler
and declaring
      | edu.someschool.its.cas.MySpecialHandler here, or you might use one of the handlers
provided in the adaptors modules.
     +-->
    <bean


/>
  </list>
</property>
```

With the following (Make sure to set the host, port and context with the values corresponding to your portal). Also available in `GATEIN_SSO_HOME/cas/plugin/WEB-INF/deployerConfigContext.xml`.

```
<!--
 | Whereas CredentialsToPrincipalResolvers identify who it is some Credentials might
authenticate,
 | AuthenticationHandlers actually authenticate credentials.  Here we declare the
AuthenticationHandlers that
| authenticate the Principals that the CredentialsToPrincipalResolvers identified.  CAS will
try these handlers in turn
 | until it finds one that both supports the Credentials presented and succeeds in
authenticating.
+-->
<property name="authenticationHandlers">
  <list>
    <!--
     | This is the authentication handler that authenticates services by means of callback via
SSL, thereby validating
     | a server side SSL certificate.
     +-->
                                                                          <bean
class="org.jasig.cas.authentication.handler.support.HttpBasedServiceCredentialsAuthenticationHandler"
         p:httpClient-ref="httpClient" />
    <!--
       | This is the authentication handler declaration that every CAS deployer will need to
change before deploying CAS
```

```
        | into production.  The default SimpleTestUsernamePasswordAuthenticationHandler
authenticates UsernamePasswordCredentials
        | where the username equals the password.  You will need to replace this with an
AuthenticationHandler that implements your
      | local authentication strategy.  You might accomplish this by coding a new such handler
and declaring
      | edu.someschool.its.cas.MySpecialHandler here, or you might use one of the handlers
provided in the adaptors modules.
     +-->
    <!-- Integrates with the Gatein Authentication Service to perform authentication -->
    <!--
        | Note: Modify the Plugin Configuration based on the actual information of a GateIn
instance.
        | The instance can be anywhere on the internet...Not necessarily on localhost where
CAS is running
     +-->
    <bean class="org.gatein.sso.cas.plugin.AuthenticationPlugin">
      <property name="gateInHost"><value>localhost</value></property>
      <property name="gateInPort"><value>8080</value></property>
      <property name="gateInContext"><value>portal</value></property>
    </bean>
  </list>
</property>
```

3. Copy `GATEIN_SSO_HOME/cas/plugin/WEB-INF/lib/sso-cas-plugin-<VERSION>.jar` and `GATEIN_SSO_HOME/cas/plugin/WEB-INF/lib/commons-httpclient-<VERSION>.jar` into the `CAS_HOME/cas-server-webapp/src/main/webapp/WEB-INF/lib` created directory.

4. Get an installation of Tomcat and extract it into a suitable location (which will be called `TOMCAT_HOME` for these instructions).

   Change the default port to avoid a conflict with the default GateIn 3.2 (for testing purposes). Edit `TOMCAT_HOME/conf/server.xml` and replace the 8080 port with 8888.

   ### Note

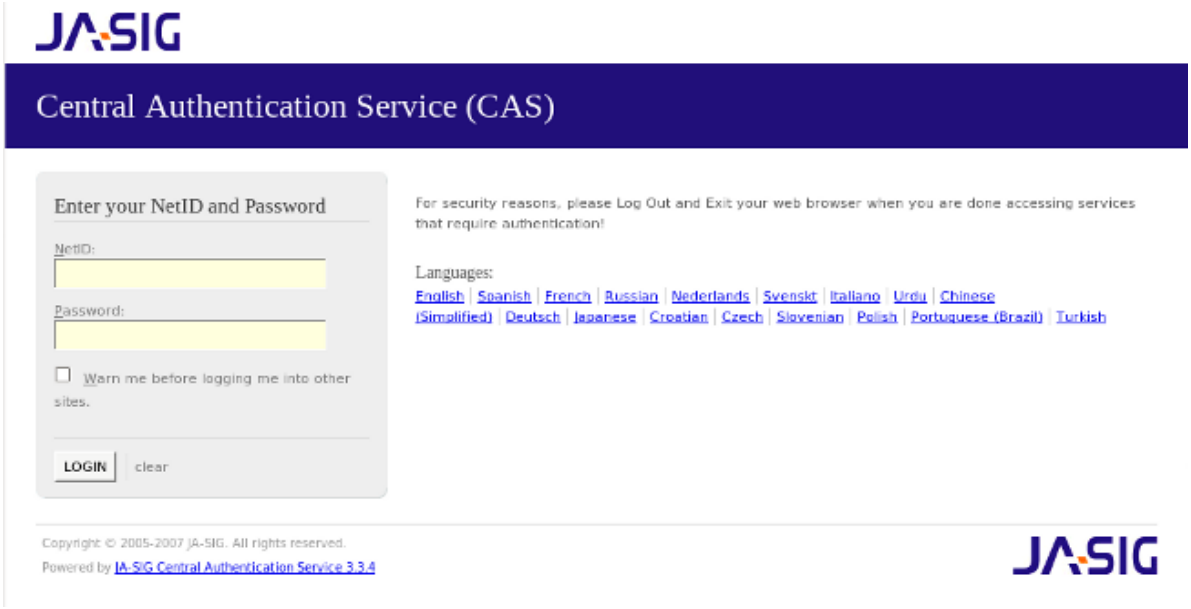
   If GateIn 3.2 is running on the same machine as Tomcat, other ports need to be changed in addition to 8080 to avoid port conflicts. They can be changed to any free port. For example, you can change admin port from 8005 to 8805, and AJP port from 8009 to 8809.

5. Go to `CAS_HOME/cas-server-webapp` and execute the command:

```
mvn install
```

6. Copy `CAS_HOME/cas-server-webapp/target/cas.war` into `TOMCAT_HOME/webapps`.

   Tomcat should start and be accessible at *http://localhost:8888/cas*. At this stage, the login will not be available.



Set up the CAS client

1. Copy all libraries from `GATEIN_SSO_HOME/cas/gatein.ear/lib` into `JBOSS_HOME/server/default/deploy/gatein.ear/lib` (Or in Tomcat, into `$GATEIN_HOME/lib`)

2. - In JBoss AS, edit `gatein.ear/META-INF/gatein-jboss-beans.xml` and uncomment on this section:

```
<authentication>
  <login-module code="org.gatein.sso.agent.login.SSOLoginModule" flag="required">
  </login-module>
        <login-module     code="org.exoplatform.services.security.j2ee.JbossLoginModule"
 flag="required">
    <module-option name="portalContainerName">portal</module-option>
    <module-option name="realmName">gatein-domain</module-option>
  </login-module>
</authentication>
```

   - In Tomcat, edit `GATEIN_HOME/conf/jaas.conf`, uncomment on this section and comment other parts:

```
org.gatein.sso.agent.login.SSOLoginModule required;
org.exoplatform.services.security.j2ee.TomcatLoginModule required
```

```
portalContainerName=portal
realmName=gatein-domain;
```

3. The installation can be tested at this point:

    1. Start (or restart) GateIn 3.2, and (assuming the CAS server on Tomcat is running) direct your browser to *http://localhost:8888/cas*.
    2. Login with the username `root` and the password `gtn` (or any account created through the portal).

Redirect to CAS

To utilize the Central Authentication Service, GateIn 3.2 needs to redirect all user authentication to the CAS server.

Information about where the CAS is hosted must be properly configured within the GateIn 3.2 instance. The required configuration is done by modifying three files:

- In the `gatein.ear/web.war/groovy/groovy/webui/component/UIBannerPortlet.gtml` file, modify the 'Sign In' link as follows:

```
<!--
<a class="Login" onclick="$signInAction"><%=_ctx.appRes("UILoginForm.label.Signin")%></
a>
-->
<a class="Login" href="/portal/sso"><%=_ctx.appRes("UILoginForm.label.Signin")%></a>
```

- In the `gatein.ear/web.war/groovy/portal/webui/component/UILogoPortlet.gtmpl` file, modify the 'Sign In' link as follows:

```
<!--
<a onclick="$signInAction"><%=_ctx.appRes("UILogoPortlet.action.signin")%></a>
-->
<a href="/portal/sso"><%=_ctx.appRes("UILogoPortlet.action.signin")%></a>
```

- Replace the entire contents of `gatein.ear/02portal.war/login/jsp/login.jsp` with:

```
<html>
  <head>
    <script type="text/javascript">
     window.location = '/portal/sso';
    </script>
  </head>
  <body>
  </body>
</html>
```

- Add the following filters at the top of the filter chain in `gatein.ear/02portal.war/WEB-INF/web.xml`:

```
 <filter>
  <filter-name>LoginRedirectFilter</filter-name>
  <filter-class>org.gatein.sso.agent.filter.LoginRedirectFilter</filter-class>
  <init-param>
    <!-- This should point to your SSO authentication server -->

    <param-name>LOGIN_URL</param-name>
    <!--
       If casRenewTicket param value of InitiateLoginServlet is:  not specified or false
    -->
          <param-value>http://localhost:8888/cas/login?service=http://localhost:8080/portal/private/
classic</param-value>
    <!--
       If casRenewTicket param value of InitiateLoginServlet is : true
    -->
    <!--
    <param-value>http://localhost:8888/cas/login?service=http://localhost:8080/portal/private
    /classic&amp;renew=true</param-value>
    -->
  </init-param>
 </filter>
 <filter>
  <filter-name>CASLogoutFilter</filter-name>
  <filter-class>org.gatein.sso.agent.filter.CASLogoutFilter</filter-class>
  <init-param>
```

```
    <!-- This should point to your JOSSO authentication server -->

   <param-name>LOGOUT_URL</param-name>
    <param-value>http://localhost:8888/cas/logout</param-value>

  </init-param>
 </filter>

 <!-- Mapping the filters at the very top of the filter chain -->
 <filter-mapping>
   <filter-name>LoginRedirectFilter</filter-name>
   <url-pattern>/*</url-pattern>
 </filter-mapping>
 <filter-mapping>
   <filter-name>CASLogoutFilter</filter-name>
   <url-pattern>/*</url-pattern>
 </filter-mapping>
```

- Replace the `InitiateLoginServlet` declaration in `gatein.ear/02portal.war/WEB-INF/web.xml` with:

```
<servlet>
  <servlet-name>InitiateLoginServlet</servlet-name>
  <servlet-class>org.gatein.sso.agent.GenericSSOAgent</servlet-class>
  <init-param>
    <param-name>ssoServerUrl</param-name>
    <param-value>http://localhost:8888/cas</param-value>
  </init-param>
  <init-param>
    <param-name>casRenewTicket</param-name>
    <param-value>false</param-value>
  </init-param>
</servlet>
```

Once these changes have been made, all links to the user authentication pages will redirect to the CAS centralized authentication form.

JOSSO

This Single-Sign-On plugin enables the seamless integration between GateIn 3.2 and the JOSSO Single-Sign-On Framework. Details about JOSSO can be found *here* [http://www.josso.org].

Setting up this integration consists of two steps: installing/configuring a JOSSO server, and setting up the portal to use the JOSSO server.

JOSSO server

This section describes how to set up the JOSSO server to authenticate against the GateIn 3.2 login module.

In this example, the JOSSO server will be installed on Tomcat.

Obtaining JOSSO

JOSSO can be downloaded from *http://sourceforge.net/projects/josso/files/*. Use the package that embeds Apache Tomcat.

Once downloaded, extract the package into what will be called `JOSSO_HOME` in this example.

The steps described later are only correct in case of JOSSO v.1.8.1.

Modifying the JOSSO server

1. Copy the files from `GATEIN_SSO_HOME/josso/plugin` into the Tomcat directory (`JOSSO_HOME`).

   This action should replace or add the following files to the `JOSSO_HOME/webapps/josso/WEB-INF/lib` directory:

   - `JOSSO_HOME/lib/josso-gateway-config.xml`
   - `JOSSO_HOME/lib/josso-gateway-gatein-stores.xml`

   and

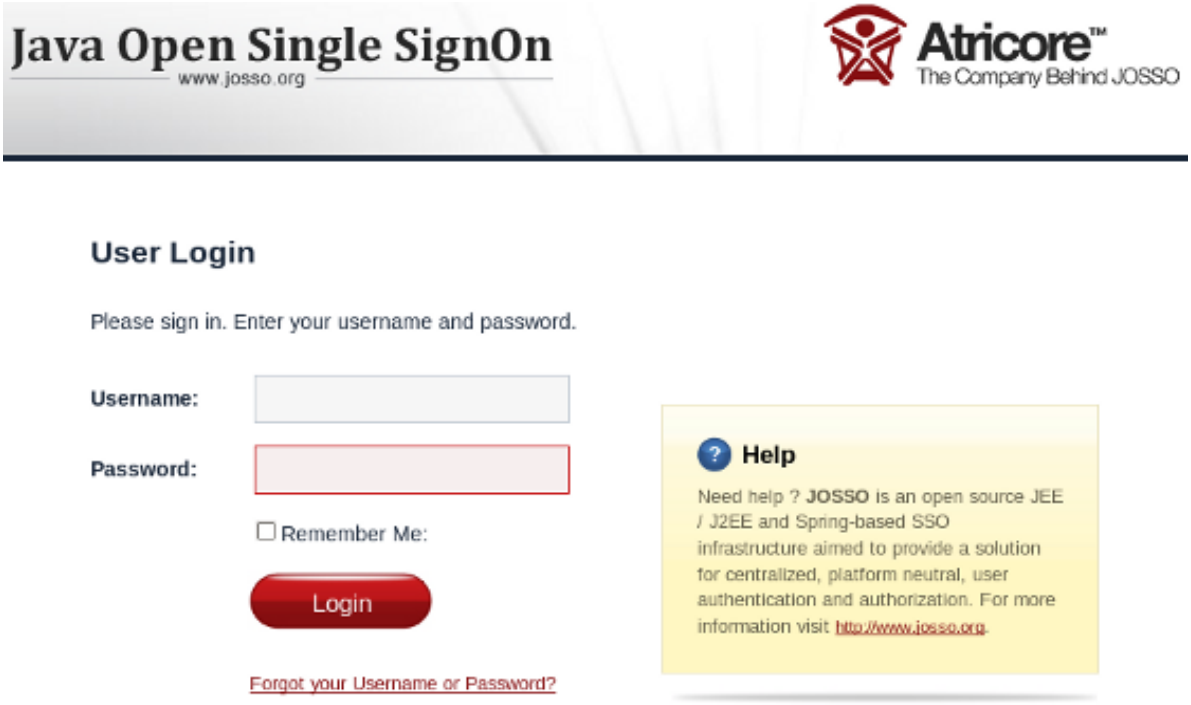
   - `JOSSO_HOME/webapps/josso/WEB-INF/classes/gatein.properties`

2. Edit `TOMCAT_HOME/conf/server.xml` and replace the 8080 port with 8888 to change the default Tomcat port and avoid a conflict with the default GateIn 3.2 port (for testing purposes).

   #### Port Conflicts

   If GateIn 3.2 is running on the same machine as Tomcat, other ports need to be changed in addition to 8080 to avoid port conflicts. They can be changed to any free port. For example, you can change the admin port from 8005 to 8805, and AJP port from 8009 to 8809.

3. Tomcat should now start and allow access to *http://localhost:8888/josso/signon/login.do*; but at this stage, login will not be available.



Set up the JOSSO client

1. Copy the library files from `GATEIN_SSO_HOME/josso/gatein.ear/lib` into `gatein.ear/lib` (or into `GATEIN_HOME/lib` if GateIn 3.2 is running in Tomcat)

2. Copy the file `GATEIN_SSO_HOME/josso/gatein.ear/portal.war/WEB-INF/classes/josso-agent-config.xml` into `gatein.ear/02portal.war/WEB-INF/classes` (or into `GATEIN_HOME/webapps/portal.war/WEB-INF/classes`, or `GATEIN_HOME/conf` if GateIn 3.2 is running in Tomcat)

3. - In JBoss AS, edit `gatein.ear/META-INF/gatein-jboss-beans.xml` and uncomment on this section:

```
<authentication>
  <login-module code="org.gatein.sso.agent.login.SSOLoginModule" flag="required">
  </login-module>
        <login-module     code="org.exoplatform.services.security.j2ee.JbossLoginModule"
 flag="required">
    <module-option name="portalContainerName">portal</module-option>
    <module-option name="realmName">gatein-domain</module-option>
  </login-module>
</authentication>
```

- In Tomcat, edit `GATEIN_HOME/conf/jaas.conf`, uncomment on this section and comment other parts:

```
org.gatein.sso.agent.login.SSOLoginModule required;
org.exoplatform.services.security.j2ee.TomcatLoginModule requiredtm
portalContainerName=portal
realmName=gatein-domain;
```

4. The installation can be tested at this point.

    1. Start (or restart) GateIn 3.2, and (assuming the JOSSO server on Tomcat is running) direct your browser to *http://localhost:8888/josso/signon/login.do*.
    2. Login with the username `root` and the password `gtn` or any account created through the portal.

Set up the portal to redirect to JOSSO

The next part of the process redirects all user authentication to the JOSSO server.

Information about where the JOSSO server is hosted must be properly configured within the GateIn 3.2 instance. The required configuration is done by modifying four files:

- In the `gatein.ear/web.war/groovy/groovy/webui/component/UIBannerPortlet.gtml` file, modify the 'Sign In' link as follows:

```
<!--
<a class="Login" onclick="$signInAction"><%=_ctx.appRes("UILoginForm.label.Signin")%></
a>
-->
<a class="Login" href="/portal/sso"><%=_ctx.appRes("UILoginForm.label.Signin")%></a>
```

- In the `gatein.ear/web.war/groovy/portal/webui/component/UILogoPortlet.gtmpl` file, modify the 'Sign In' link as follows:

```
<!--
```

```
  <a onclick="$signInAction"><%=_ctx.appRes("UILogoPortlet.action.signin")%></a>
  -->
  <a href="/portal/sso"><%=_ctx.appRes("UILogoPortlet.action.signin")%></a>
```

- Replace the entire contents of `gatein.ear/02portal.war/login/jsp/login.jsp` with:

```
<html>
  <head>
    <script type="text/javascript">
     window.location = '/portal/sso';
    </script>
  </head>
  <body>
  </body>
</html>
```

- Add the following filters at the top of the filter chain in `gatein.ear/02portal.war/WEB-INF/web.xml`:

```
  <filter>
      <filter-name>LoginRedirectFilter</filter-name>
      <filter-class>org.gatein.sso.agent.filter.LoginRedirectFilter</filter-class>
      <init-param>
   <!-- This should point to your SSO authentication server -->

  <param-name>LOGIN_URL</param-name>
  <param-value>http://localhost:8888/josso/signon/login.do?josso_back_to=http://localhost:8080/
portal
  /private/classic</param-value>
      </init-param>
  </filter>
  <filter>
     <filter-name>JOSSOLogoutFilter</filter-name>
       <filter-class>org.gatein.sso.agent.filter.JOSSOLogoutFilter</filter-class>

      <init-param>
         <!-- This should point to your JOSSO authentication server -->
```

```
        <param-name>LOGOUT_URL</param-name>

       <param-value>http://localhost:8888/josso/signon/logout.do</param-value>

    </init-param>
 </filter>

 <!-- filters should be placed at the very top of the filter chain -->
 <filter-mapping>
  <filter-name>LoginRedirectFilter</filter-name>
  <url-pattern>/*</url-pattern>
 </filter-mapping>
 <filter-mapping>
    <filter-name>JOSSOLogoutFilter</filter-name>
    <url-pattern>/*</url-pattern>
 </filter-mapping>
```

- Replace the `InitiateLoginServlet` declaration in `gatein.ear/02portal.war/WEB-INF/web.xml` with:

```
<servlet>
  <servlet-name>InitiateLoginServlet</servlet-name>
  <servlet-class>org.gatein.sso.agent.GenericSSOAgent</servlet-class>
  <init-param>
    <param-name>ssoServerUrl</param-name>
    <param-value>http://localhost:8888/josso/signon/login.do</param-value>
  </init-param>
</servlet>
```

- Remove the `PortalLoginController` servlet declaration and mapping in `gatein.ear/02portal.war/WEB-INF/web.xml`

From now on, all links redirecting to the user authentication pages will redirect to the JOSSO centralized authentication form.

OpenSSO - The Open Web SSO project

This Single-Sign-On plugin enables the seamless integration between GateIn 3.2 and the OpenSSO Single Sign On Framework. Details about OpenSSO can be found *here* [https://opensso.dev.java.net/].

Setting up this integration consists of two steps: installing/configuring an OpenSSO server, and setting up the portal to use the OpenSSO server.

OpenSSO server

This section details the setup of OpenSSO server to authenticate against the GateIn 3.2 login module.

In this example, the OpenSSO server will be installed on Tomcat.

Obtaining OpenSSO

OpenSSO can be downloaded from *http://download.oracle.com/otn/nt/middleware/11g/oracle_opensso_80U2.zip*.

Once downloaded, extract the package into a suitable location. This location will be referred to as `OPENSSO_HOME` in this example.

Modifying the OpenSSO server

To configure the web server as desired, it is simpler to directly modify the sources.

The first step is to add the GateIn 3.2 Authentication Plugin:

The plugin makes secure authentication callbacks to a RESTful service installed on the remote GateIn 3.2 server to authenticate a user.

In order for the plugin to function correctly, it needs to be properly configured to connect to this service. This configuration is done via the `opensso.war/config/auth/default/AuthenticationPlugin.xml` file.

1. Obtain a copy of Tomcat and extract it into a suitable location (this location will be referred to as `TOMCAT_HOME` in this example).
2. Change the default port to avoid a conflict with the default GateIn 3.2 port (for testing purposes) by editing `TOMCAT_HOME/conf/server.xml` and replacing the 8080 port with 8888.

   #### Note

   If GateIn 3.2 is running on the same machine as Tomcat, other ports need to be changed in addition to 8080 to avoid port conflicts. They can be changed to any free port. For example, you can change the admin port from 8005 to 8805, and AJP port from 8009 to 8809.

3. Ensure the `TOMCAT_HOME/webapps/opensso/config/auth/default/AuthenticationPlugin.xml` file looks like this:

```
<?xml version='1.0' encoding="UTF-8"?>

<!DOCTYPE ModuleProperties PUBLIC "=//iPlanet//Authentication Module Properties XML
 Interface 1.0 DTD//EN"
        "jar://com/sun/identity/authentication/Auth_Module_Properties.dtd">

<ModuleProperties moduleName="AuthenticationPlugin" version="1.0" >
  <Callbacks length="2" order="1" timeout="60"
           header="GateIn OpenSSO Login" >
    <NameCallback>
      <Prompt>
  Username
      </Prompt>
    </NameCallback>
    <PasswordCallback echoPassword="false" >
      <Prompt>
  Password
      </Prompt>
    </PasswordCallback>
  </Callbacks>
</ModuleProperties>
```

4. Copy `GATEIN_SSO_HOME/opensso/plugin/WEB-INF/lib/sso-opensso-plugin-<VERSION>.jar`, `GATEIN_SSO_HOME/opensso/plugin/WEB-INF/lib/commons-httpclient-<VERSION>.jar`, and `GATEIN_SSO_HOME/opensso/plugin/WEB-INF/lib/commons-logging-<VERSION>.jar` into the Tomcat directory at `TOMCAT_HOME/webapps/opensso/WEB-INF/lib`.

5. Copy `GATEIN_SSO_HOME/opensso/plugin/WEB-INF/classes/gatein.properties` into `TOMCAT_HOME/webapps/opensso/WEB-INF/classes`.

6. Tomcat should start and be able to access *http://localhost:8888/opensso/UI/Login?realm=gatein*. Login will not be available at this point.

Configure the "gatein" realm:

1. Direct your browser to *http://localhost:8888/opensso*.
2. Create the default configuration.
3. Login as `an admin` and then go to the "Configuration" tab -> "Authentication" -> "Core" link -> add a new value and fill in the class name "org.gatein.sso.opensso.plugin.AuthenticationPlugin". This step is really important. If not, AuthenticationPlugin is not available among other the OpenSSO authentication modules.
4. Go to the "Access control" tab and create the new realm called "gatein".
5. Go to the "gatein" realm and click the "Authentication" tab. At the bottom of the "Authentication chaining" section, click "ldapService". Here, change the selection from "Datastore", which is the default module in the authentication chain, to "AuthenticationPlugin". This enables the authentication of "gatein" realm by using the GateIn REST service instead of the OpenSSO LDAP server.
6. Go to "Advanced properties" and change UserProfile from "Required" to "Dynamic". This step is needed because GateIn 3.2 users are not in the OpenSSO Datastore (LDAP server), so their profiles can not be obtained if "Required" is active. By using "Dynamic", all new users are automatically created in the OpenSSO datastore after successful authentication.

7. Increase the user privileges to allow the REST access. Go to "Access control" -> Top level realm -> "Privileges" tab -> All authenticated users, and check the last two checkboxes:

   - Read and write access only for policy properties.
   - Read and write access to all realm and policy properties.

8. Do the same for the "gatein" realm.

Also, instead of configuring OpenSSO manually as above, you can refer to the available configuration files *here* [https://repository.jboss.org/nexus/content/groups/public/org/gatein/sso/sso-packaging/1.0.1-GA/sso-packaging-1.0.1-GA.zip].

Set up the OpenSSO client.

1. Copy all libraries from `GATEIN_SSO_HOME/opensso/gatein.ear/lib` into `JBOSS_HOME/server/default/deploy/gatein.ear/lib` (Or, in Tomcat, into `GATEIN_HOME/lib`)

2. - In JBoss AS, edit gatein.ear/META-INF/gatein-jboss-beans.xml and uncomment on this section

```
<authentication>
  <login-module code="org.gatein.sso.agent.login.SSOLoginModule" flag="required">
  </login-module>
        <login-module     code="org.exoplatform.services.security.j2ee.JbossLoginModule"
 flag="required">
    <module-option name="portalContainerName">portal</module-option>
    <module-option name="realmName">gatein-domain</module-option>
  </login-module>
</authentication>
```

   - If you are running GateIn 3.2 in Tomcat, edit $GATEIN_HOME/conf/jaas.conf, uncomment on this section and comment other parts:

```
org.gatein.sso.agent.login.SSOLoginModule required;
org.exoplatform.services.security.j2ee.TomcatLoginModule required
portalContainerName=portal
realmName=gatein-domain;
```

   At this point, the installation can be tested:

   1. Access GateIn 3.2 by going to *http://localhost:8888/opensso/UI/Login?realm=gatein* (assuming that the OpenSSO server using Tomcat is still running).

2. Login with the username `root` and the password `gtn` or any account created through the portal.

Set up the portal to redirect to OpenSSO

The next part of the process is to redirect all user authentication to the OpenSSO server.

Information about where the OpenSSO server is hosted must be properly configured within the Enterprise Portal Platform instance. The required configuration is done by modifying three files:

- In the `gatein.ear/web.war/groovy/groovy/webui/component/UIBannerPortlet.gtml` file, modify the 'Sign In' link as follows:

```
<!--
<a class="Login" onclick="$signInAction"><%=_ctx.appRes("UILoginForm.label.Signin")%></
a>
-->
<a class="Login" href="/portal/sso"><%=_ctx.appRes("UILoginForm.label.Signin")%></a>
```

- In the `gatein.ear/web.war/groovy/portal/webui/component/UILogoPortlet.gtmpl` file, modify the 'Sign In' link as follows:

```
<!--
<a onclick="$signInAction"><%=_ctx.appRes("UILogoPortlet.action.signin")%></a>
-->
<a href="/portal/sso"><%=_ctx.appRes("UILogoPortlet.action.signin")%></a>
```

- Replace the entire contents of `gatein.ear/02portal.war/login/jsp/login.jsp` with:

```
<html>
  <head>
    <script type="text/javascript">
     window.location = '/portal/sso';
    </script>
```

```
 </head>
 <body>
 </body>
</html>
```

- Add the following filters at the top of the filter chain in `gatein.ear/02portal.war/WEB-INF/web.xml`:

```
 <filter>
    <filter-name>LoginRedirectFilter</filter-name>
    <filter-class>org.gatein.sso.agent.filter.LoginRedirectFilter</filter-class>
    <init-param>
<!-- This should point to your SSO authentication server -->

<param-name>LOGIN_URL</param-name>
            <param-value>http://localhost:8888/opensso/UI/Login?realm=gatein&amp;goto=http://
localhost:8080
/portal/private/classic</param-value>
    </init-param>
 </filter>
 <filter>
     <filter-name>OpenSSOLogoutFilter</filter-name>

     <filter-class>org.gatein.sso.agent.filter.OpenSSOLogoutFilter</filter-class>

    <init-param>
      <!-- This should point to your OpenSSO authentication server -->

       <param-name>LOGOUT_URL</param-name>

      <param-value>http://localhost:8888/opensso/UI/Logout</param-value>

    </init-param>
 </filter>

 <!-- place the filters at the top of the filter chain -->
 <filter-mapping>
  <filter-name>LoginRedirectFilter</filter-name>
  <url-pattern>/*</url-pattern>
 </filter-mapping>
 <filter-mapping>
```

```
    <filter-name>OpenSSOLogoutFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

- Replace the `InitiateLoginServlet` declaration in `gatein.ear/02portal.war/WEB-INF/web.xml` with:

```
<servlet>
  <servlet-name>InitiateLoginServlet</servlet-name>
  <servlet-class>org.gatein.sso.agent.GenericSSOAgent</servlet-class>
  <init-param>
    <param-name>ssoServerUrl</param-name>
    <param-value>http://localhost:8888/opensso</param-value>
  </init-param>
  <init-param>
    <param-name>ssoCookieName</param-name>
    <param-value>iPlanetDirectoryPro</param-value>
  </init-param>
</servlet>
```

From now on, all links redirecting to the user authentication pages will redirect to the OpenSSO centralized authentication form.

SPNEGO

SPNEGO (Simple and Protected GSSAPI Negotiation Mechanism) is used to authenticate transparently through the web browser after the user has been authenticated when logging in.

See the following typical usecase:

1. The user logs into the desktop, such as a Windows machine.
2. The desktop login is governed by the Active Directory domain.
3. Next, the user opens the browser (IE/Firefox) to access a web application (that uses JBoss Negotiation) hosted on JBoss EPP.
4. The Browser transfers the desktop sign-on information to the web application.
5. JBoss EAP/AS uses the background GSS messages with the Active Directory (or any Kerberos server) to validate the user.
6. The User has a seamless SSO into the web application.

SPNEGO Server Configuration

Note

Information stated here only describes basic steps for you to configure the SPNEGO server. If you are already familiar with SPNEGO, jump to the *the section called "GateIn 3.2 Configuration"* to see how to integrate SPNEGO with GateIn.

1. Correct the setup of network on the machine. For example, if you are using the "server.local.network" domain as your machine where Kerberos and GateIn 3.2 are localed, add the line containing the machine's IP address to the **/etc/host** file.

```
192.168.1.88  server.local.network
```

Note

It is not recommended you use loopback addresses.

2. Install Kerberos with these packages: krb5-admin-server, krb5-kdc, krb5-config, krb5-user, krb5-clients, and krb5-rsh-server.
3. Edit the Kerberos configuration file at **/etc/krb5.config**, including:

   - Uncomment on these lines:

```
default_tgs_enctypes = des3-hmac-sha1
default_tkt_enctypes = des3-hmac-sha1
permitted_enctypes = des3-hmac-sha1
```

   - Add **local.network** as a default realm and it is also added to the list of realms and remove the remains of realms. The content looks like:

```
[libdefaults]
    default_realm = LOCAL.NETWORK

# The following krb5.conf variables are only for MIT Kerberos.
    krb4_config = /etc/krb.conf
    krb4_realms = /etc/krb.realms
```

```
    kdc_timesync = 1
    ccache_type = 4
    forwardable = true
    proxiable = true

# The following encryption type specification will be used by MIT Kerberos
# if uncommented.  In general, the defaults in the MIT Kerberos code are
# correct and overriding these specifications only serves to disable new
# encryption types as they are added, creating interoperability problems.
#
# Thie only time when you might need to uncomment these lines and change
# the enctypes is if you have local software that will break on ticket
# caches containing ticket encryption types it doesn't know about (such as
# old versions of Sun Java).

    default_tgs_enctypes = des3-hmac-sha1
    default_tkt_enctypes = des3-hmac-sha1
    permitted_enctypes = des3-hmac-sha1

# The following libdefaults parameters are only for Heimdal Kerberos.
    v4_instance_resolve = false
    v4_name_convert = {
        host = {
            rcmd = host
            ftp = ftp
        }
        plain = {
            something = something-else
        }
    }
    fcc-mit-ticketflags = true

[realms]
    LOCAL.NETWORK = {
        kdc = server.local.network
        admin_server = server.local.network
    }

[domain_realm]
    .local.network = LOCAL.NETWORK
    local.network = LOCAL.NETWORK

[login]
    krb4_convert = true
```

```
    krb4_get_tickets = false
```

4. Edit the KDC configuraton file at **/etc/krb5kdc/kdc.conf** that looks like.

```
[kdcdefaults]
   kdc_ports = 750,88

[realms]
   LOCAL.NETWORK = {
      database_name = /home/gatein/krb5kdc/principal
      admin_keytab = FILE:/home/gatein/krb5kdc/kadm5.keytab
      acl_file = /home/gatein/krb5kdc/kadm5.acl
      key_stash_file = /home/gatein/krb5kdc/stash
      kdc_ports = 750,88
      max_life = 10h 0m 0s
      max_renewable_life = 7d 0h 0m 0s
      master_key_type = des3-hmac-sha1
     supported_enctypes = aes256-cts:normal arcfour-hmac:normal des3-hmac-sha1:normal
 des-cbc-crc:normal des:normal des:v4 des:norealm des:onlyrealm des:afs3
      default_principal_flags = +preauth
   }

[logging]
      kdc = FILE:/home/gatein/krb5logs/kdc.log
      admin_server = FILE:/home/gatein/krb5logs/kadmin.log
```

- Create krb5kdc and krb5logs directory for Kerberos database as shown in the configuration file above.
- Next, create a KDC database using the following command.

```
sudo krb5_newrealm
```

- Start the KDC and Kerberos admin servers using these commands:

```
sudo /etc/init.d/krb5-kdc restart
sudo /etc/init.d/krb-admin-server restart
```

5. Add Principals and create Keys.

   - Start an interactive 'kadmin' session and create the necessary Principals.

```
sudo kadmin.local
```

   - Add the GateIn 3.2 machine and keytab file that need to be authenticated.

```
addprinc -randkey HTTP/server.local.network@LOCAL.NETWORK
ktadd HTTP/server.local.network@LOCAL.NETWORK
```

   - Add the default GateIn 3.2 user accounts and enter the password for each created user that will be authenticated.

```
addprinc john
addprinc demo
addprinc root
```

6. Test your changed setup by using the command.

```
kinit -A demo
```

   - If the setup works well, you are required to enter the password created for this user in Step 5.
   - If you want to login with another user, use this command.

```
kdestroy
```

GateIn 3.2 Configuration

GateIn 3.2 uses JBoss Negotiation to enable SPNEGO-based desktop SSO for the portal. Here are the steps to integrate SPNEGO with GateIn 3.2.

1. Activate the Host authentication under the **conf/login-config.xml** file adding the following host login module:

```
<application-policy>
<authentication>
<login-module>
<module-option>true</module-option>

<module-option>true</module-option>

<module-option>HTTP/server.local.network@LOCAL.NETWORK</module-option>

<module-option>/etc/krb5.keytab</module-option>

<module-option>true</module-option>

<module-option>true</module-option>
   </login-module>
 </authentication>
 </application-policy>
```

   The 'keyTab' value should point to the keytab file that was generated by the kadmin kerberos tool. See the *the section called "SPNEGO Server Configuration"* section for more details.

2. Extend the core authentication mechanisms to support SPNEGO under **deployers/jbossweb.deployer/META-INF/war-deployers-jboss-beans.xml** by adding the 'SPNEGO' authenticators property.

```
<deployment xmlns="urn:jboss:bean-deployer:2.0">
```

```
<property name="authenticators">
                    <map          class="java.util.Properties"          keyClass="java.lang.String"
 valueClass="java.lang.String">
   <entry>
    <key>BASIC</key>
    <value>org.apache.catalina.authenticator.BasicAuthenticator</value>
   </entry>
   <entry>
    <key>CLIENT-CERT</key>
    <value>org.apache.catalina.authenticator.SSLAuthenticator</value>
   </entry>
   <entry>
    <key>DIGEST</key>
    <value>org.apache.catalina.authenticator.DigestAuthenticator</value>
   </entry>
   <entry>
    <key>FORM</key>
    <value>org.apache.catalina.authenticator.FormAuthenticator</value>
   </entry>
   <entry>
    <key>NONE</key>
    <value>org.apache.catalina.authenticator.NonLoginAuthenticator</value>
   </entry>

   <!-- Add this entry -->
   <entry>
    <key>SPNEGO</key>
    <value>org.jboss.security.negotiation.NegotiationAuthenticator</value>
   </entry>
  </map>
</property>
```

3. Add the GateIn 3.2 SSO module binaries by copying **$GATEIN_SSO_HOME/spnego/gatein.ear/lib/sso-agent.jar**, and **$GATEIN_SSO_HOME/spnego/gatein.ear/lib/sso-spnego.jar** to the **deploy/gatein.ear/lib** directory.

4. Modify the **deploy/gatein.ear/META-INF/gatein-jboss-beans.xml** file as below, then comment on other parts.

```
<login-module code="org.gatein.sso.spnego.SPNEGOLoginModule" flag="required">
  <module-option name="password-stacking">useFirstPass</module-option>
  <module-option name="serverSecurityDomain">host</module-option>
```

```
</login-module>
<login-module code="org.gatein.sso.agent.login.SPNEGORolesModule" flag="required">
  <module-option name="password-stacking">useFirstPass</module-option>
  <module-option name="portalContainerName">portal</module-option>
  <module-option name="realmName">gatein-domain</module-option>
</login-module>
```

5. Modify **gatein.ear/02portal.war/WEB-INF/web.xml** as below.

```
<!--
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>gatein-domain</realm-name>
  <form-login-config>
    <form-login-page>/initiatelogin</form-login-page>
    <form-error-page>/errorlogin</form-error-page>
  </form-login-config>
</login-config>
-->

<login-config>
  <auth-method>SPNEGO</auth-method>
  <realm-name>SPNEGO</realm-name>
</login-config>
```

   This integrates SPNEGO support into the Portal web archive by switching the authentication mechanism from the default "FORM"-based to "SPNEGO"-based authentication.

6. Integrate the request pre-processing needed for SPNEGO via filters by adding the following filters to the **web.xml** at the top of the Filter chain.

```
<filter>
  <filter-name>LoginRedirectFilter</filter-name>
  <filter-class>org.gatein.sso.agent.filter.LoginRedirectFilter</filter-class>
  <init-param>
  <!-- This should point to your SSO authentication server -->
    <param-name>LOGIN_URL</param-name>
    <param-value>/portal/private/classic</param-value>
  </init-param>
</filter>
```

```
<filter>
  <filter-name>SPNEGOFilter</filter-name>
  <filter-class>org.gatein.sso.agent.filter.SPNEGOFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>LoginRedirectFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>SPNEGOFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

7. Start the GateIn 3.2 portal using the command below.

```
sudo ./run.sh -Djava.security.krb5.realm=LOCAL.NETWORK -Djava.security.krb5.kdc=server.local.network -c PROFILE -b server.local.network
```

The PROFILE parameter in the above command should be replaced with the server profile modified in the above configuration. For example, if you are configuring the default profile, your command will be.

```
sudo ./run.sh -Djava.security.krb5.realm=LOCAL.NETWORK -Djava.security.krb5.kdc=server.local.network -c default -b server.local.network
```

8. Login to Kerberos with the command.

```
kinit -A demo
```

You should be able to click the 'Sign In' link on the GateIn 3.2 portal and the 'demo' user from the GateIn 3.2 portal should be automatically logged in.

Clients

After performing all configurations above, you need to enable the **Negotiate authentication** of Firefox in clients so that clients could be authenticated by GateIn 3.2 as follows:

1. Start Firefox, then enter the command: **about:config** into the address field.
2. Enter **network.negotiate-auth** and set the value as below:

```
network.negotiate-auth.allow-proxies = true
network.negotiate-auth.delegation-uris = .local.network
network.negotiate-auth.gsslib (no-value)
network.negotiate-auth.trusted-uris = .local.network
network.negotiate-auth.using-native-gsslib = true
```

Web Services for Remote Portlets (WSRP)

Introduction

The Web Services for Remote Portlets specification defines a web service interface for accessing and interacting with presentation-oriented web services. It has been produced through the efforts of the Web Services for Remote Portlets (WSRP) OASIS Technical Committee. It is based on the requirements gathered and on the concrete proposals made to the committee.

Scenarios motivating the WSRP functionality include:

- Content hosts, such as portal servers, providing Portlets as presentation-oriented web services that can be used by aggregation engines.
- Aggregating frameworks, including portal servers, consuming presentation-oriented web services offered by content providers and integrating them into the framework.

More information on WSRP can be found on the *official website for WSRP* [http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp]. It is suggested that you read the *primer* [http://www.oasis-open.org/committees/download.php/10539/wsrp-primer-1.0.html] for a good, and albeit technical overview of WSRP.

Level of support in GateIn 3.2

The WSRP Technical Committee defined *WSRP Use Profiles* [http://www.oasis-open.org/committees/download.php/3073] to help with the WSRP interoperability. You can refer to terms defined in that document in this section.

GateIn provides a Simple level of support for our WSRP Producer except that out-of-band registration is not currently handled. eXo Platform supports the in-band registration and persistent local state (which are defined at the Complex level).

On the Consumer side, GateIn provides a Medium level of support for WSRP, except that eXo Platform only handles the HTML markup (as GateIn itself does not handle other markup types). eXo Platform supports the explicit portlet cloning and provides the full support the PortletManagement interface.

As far as caching goes, eXo Platform has Level 1 Producer and Consumer. eXo Platform supports the Cookie handling properly on the Consumer and our Producer that requires initialization of cookies. eXo Platform does not support custom window states or modes. However, eXo Platform supports CSS on both the Producer (though it's more a function of the portlets than inherent Producer capability) and Consumer.

While eXo Platform provides a complete implementation of WSRP 1.0, you need to go through the *Conformance statements* [http://www.oasis-open.org/committees/download.php/6018] and perform more interoperability testing (an area that needs to be better supported by the WSRP Technical Committee and Community at large).

Note

As of version 3.2 of GateIn, WSRP is only activated and supported when GateIn is deployed on the JBoss Application Server.

Deploying GateIn's WSRP services

GateIn provides a complete support of WSRP 1.0 standard interfaces and offers services to both consumers and producers. The WSRP support is provided by the following files, assuming `$GATEIN_HOME` is where GateIn has been installed, `$WSRP_VERSION` (at the time of the writing, it was 1.1.0-GA) is the version of the WSRP component and `$PORTAL_VERSION` (at the time of the writing, it was 3.0.1-GA) is the current GateIn version:

- `$GATEIN_HOME/wsrp-admin-gui.war`, which contains the WSRP Configuration portlet with which you can configure consumers to access remote servers and how the WSRP producer is configured.
- `$GATEIN_HOME/wsrp-producer.war`, which contains the WSRP producer web application.
- `$GATEIN_HOME/lib/wsrp-common-$WSRP_VERSION.jar`, which contains common classes needed by the different WSRP libraries.
- `$GATEIN_HOME/lib/wsrp-consumer-$WSRP_VERSION.jar`, which contains the WSRP consumer.
- `$GATEIN_HOME/lib/wsrp-integration-api-$WSRP_VERSION.jar`, which contains the API classes needed to integrate the WSRP component into portals.
- `$GATEIN_HOME/lib/wsrp-producer-lib-$WSRP_VERSION.jar`, which contains the classes needed by the WSRP producer.
- `$GATEIN_HOME/lib/wsrp-wsrp1-ws-$WSRP_VERSION.jar`, which contains the generated JAX-WS classes for WSRP v.1.
- `$GATEIN_HOME/lib/wsrp-wsrp2-ws-$WSRP_VERSION.jar`, which contains the generated JAX-WS classes for WSRP version 2.
- `$GATEIN_HOME/lib/gatein.portal.component.wsrp-$PORTAL_VERSION.jar`, which contains the code to integrate the WSRP service into GateIn.

If you are not going to use WSRP in GateIn, you can remove `$GATEIN_HOME/lib/gatein.portal.component.wsrp-$PORTAL_VERSION.jar` from your GateIn distribution to easily deactivate the *WSRP support* [http://community.jboss.org/wiki/WSRPserviceremovalprocedure]. Of course, if you want to trim your installation, you can also remove all the files mentioned above.

If you consider the WSRP use when running GateIn on a non-default port or hostname

JBoss WS (the web service stack that GateIn uses), you need to pay attention to details of updating the port and host name used in WSDL. See the *JBoss WS user guide on that subject* [http://community.jboss.org/wiki/JBossWS-UserGuide#Configuration] for more details.

Of course, if you have modified the host name and port on which your server runs, you will need to update the configuration for the consumer used to consume GateIn's 'self' producer. Please refer to the *???* to learn how to do so.

Considerations to use WSRP with SSL

It is possible to use WSRP over SSL for secure exchange of data. Please refer to the *instructions* [http://community.jboss.org/wiki/ConfiguringWSRPforuseoverSSL] on how to do from *GateIn's wiki* [http://community.jboss.org/wiki/GateIn].

Making a portlet remotable

Note

Only JSR-286 (Portlet 2.0) portlets can be made remotable as the mechanism to expose a portlet to WSRP relies on the JSR-286-only functionality.

GateIn does **NOT** expose local portlets for consumption by remote WSRP consumers by default. To make a portlet remotely available, it must be made "remotable" by marking it as such in the associated `portlet.xml`. This is accomplished by using a specific `org.gatein.pc.remotable container-runtime-option`. Setting its value to `true` makes the portlet available for the remote consumption, while setting its value to `false` will not publish it remotely. As specifying the remotable status for a portlet is optional, you do not need to do anything if you do not need your portlet to be available remotely.

In the following example, the "BasicPortlet" portlet is specified as being remotable.

Example 7.1.

```
<?xml version="1.0" standalone="yes"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd http://
java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
        version="2.0">
<portlet-app>
  <portlet>
```

```
    <portlet-name>BasicPortlet</portlet-name>

    ...

    <container-runtime-option>
      <name>org.gatein.pc.remotable</name>
      <value>true</value>
    </container-runtime-option>
  </portlet>
</portlet-app>
```

It is also possible to specify that all the portlets declared within a given portlet application to be remotable by default. This is done by specifying the `container-runtime-option` at the `portlet-app` element level. Individual portlets can override that value to not be remotely exposed. Let's look at an example:

Example 7.2.

```
<?xml version="1.0" standalone="yes"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd http://
java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
          version="2.0">
<portlet-app>

  <portlet>
    <portlet-name>RemotelyExposedPortlet</portlet-name>
    ...
  </portlet>
  <portlet>
    <portlet-name>NotRemotelyExposedPortlet</portlet-name>
    ...
    <container-runtime-option>
      <name>org.gatein.pc.remotable</name>
      <value>false</value>
    </container-runtime-option>
  </portlet>

  <container-runtime-option>
    <name>org.gatein.pc.remotable</name>
    <value>true</value>
```

```
  </container-runtime-option>
</portlet-app>
```

In the example above, two portlets are defined. The `org.gatein.pc.remotable container-runtime-option` is set to `true` at the `portlet-app` level, meaning that all portlets defined in this particular portlet application are exposed remotely by GateIn's WSRP producer. Note, however, that it is possible to override the default behavior: specifying a value for the `org.gatein.pc.remotable container-runtime-option` at the `portlet` level will take precedence over the default. In the example above, the `RemotelyExposedPortlet` inherits the remotable status defined at the `portlet-app` level since it does not specify a value for the`org.gatein.pc.remotable container-runtime-option`. The`NotRemotelyExposedPortlet`, however, overrides the default behavior and is not remotely exposed. Note that in the absence of a top-level `org.gatein.pc.remotable container-runtime-option` value set to`true`, portlets are NOT remotely exposed.

Consuming GateIn's WSRP portlets from a remote Consumer

WSRP Consumers vary a lot as far as how they are configured. Most of them require you to specify the URL for the Producer's WSDL definition. Please refer to your Consumer's documentation for specific instructions. For instructions on how to do so in GateIn, please refer to *???*.

GateIn's Producer is automatically set up when you deploy a portal instance with the WSRP service. You can access the WSDL file at `http://{hostname}:{port}/wsrp-producer/v2/MarkupService?wsdl`. If you wish to use only the WSRP 1 compliant version of the producer, please use the WSDL file found at `http://{hostname}:{port}/wsrp-producer/v1/MarkupService?wsdl`. The default hostname is `localhost` and the default port is 8080.

Consuming remote WSRP portlets in GateIn

Overview

To consume the WSRP portlets exposed by a remote producer, GateIn's WSRP consumer needs to know how to access that remote producer. One can configure access to a remote producer using the WSRP Producer descriptors. Alternatively, a portlet is provided to configure the remote producers.

Once a remote producer has been configured, the portlets that it exposes are then available in the Application Registry to be added to categories and then to pages.

As a way to test the WSRP producer service and to check that the portlets that you want to expose remotely are correctly published via WSRP, a default consumer named `self`, that consumes the portlets exposed by GateIn's producer, has been configured.

Configuring a remote producer walk-through

Let's work through the steps of defining access to a remote producer so that its portlets can be consumed within GateIn. You need to configure access to the Oracle's public WSRP producer. First, examine how to do so using the configuration portlet and then show how the same result can be accomplished with a producer descriptor, though it is far easier to do so via the configuration portlet.

Note

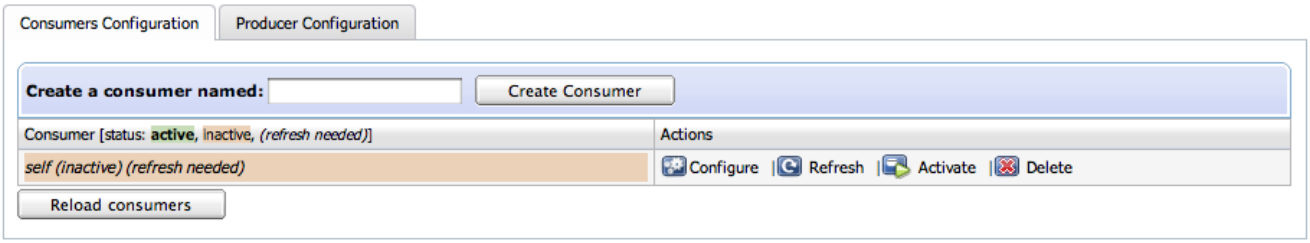
Some WSRP producers do not support the chunked encoding that is activated by default by JBoss WS. If your producer does not support the chunked encoding, your consumer will not be able to properly connect to the producer. This will manifest itself with the following error: `Caused by: org.jboss.ws.WSException: Invalid HTTP server response [503] - Service Unavailable`. Please see the GateIn's *wiki page* [http://community.jboss.org/wiki/Workaroundwhenchunkedencodingisnotsupported] for more details.

Using the configuration portlet

GateIn provides a portlet to configure access (among other functions) to remote WSRP Producers graphically. However, it is not installed by default, so the first thing you need to do is to install the WSRP configuration portlet using the Application Registry.

Use the usual procedure to log in as a Portal administrator and go to the Application Registry. With the default installment, you can just go to *http://localhost:8080/portal/login?initialURI=%2Fportal%2Fprivate%2Fclassic%2FwsrpConfigurationp&username=root&password=gtn* [http://localhost:8080/portal/login?initialURI=%2Fportal%2Fprivate%2Fclassic%2FwsrpConfiguration&username=root&password=gtn]

You should see a screen similar to:



This screen presents all the configured Consumers associated with their status and possible actions on them. A Consumer can be active or inactive. Activating a Consumer means that it is ready to act as a portlet provider. Also, note that a Consumer can be marked as required refreshing, meaning that the information held about it might not be up to date and refreshing it from the remote Producer might be a good idea. This can happen for several reasons: the service description for that remote Producer has not been fetched yet, the cached version has expired or modifications have been made to the configuration that could potentially invalidate it; thus, it is required to validate the information again.

Note

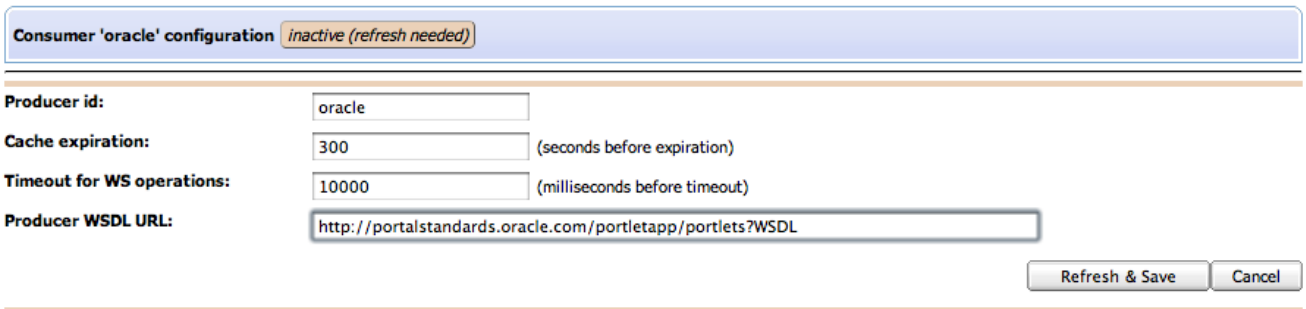
The WSRP configuration did not use to be installed by default in previous versions of GateIn. We include here the legacy instructions on how to install this portlet in case you ever need to re-install it.

Use the usual procedure to log in as a Portal administrator and go to the Application Registry. With the default install, you can just go to *http://localhost:8080/portal/login?initialURI=%2Fportal%2Fprivate%2Fclassic%2Fadministration%2Fregistry&username=root&password=gt* [http://localhost:8080/portal/login?initialURI=%2Fportal%2Fprivate%2Fclassic%2Fadministration%2Fregistry&username=root&password=gt
Add the WSRP Configuration portlet to the Administration category. If you use the Import Applications functionality, the WSRP Configuration portlet will be automatically added to the Administration category.

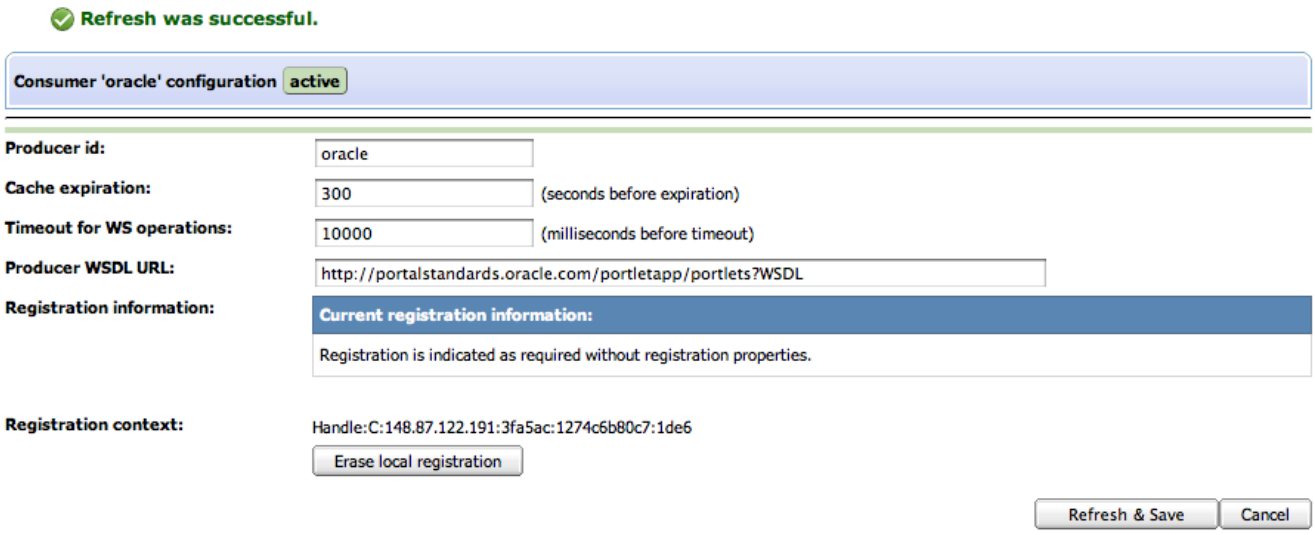
Now that the portlet is added to a category, it can be added to a page and used. We recommend adding it to the same page as the Application Registry as operations relating to WSRP and adding portlets to categories are somewhat related as we will see. Go ahead and add the WSRP Configuration portlet to the page using the standard procedure.

Next, you need create a new Consumer called `oracle`. Type "`oracle`" in the "Create a consumer named:" field, then click "Create consumer":

You should now see a form allowing you to enter/modify the information about the Consumer. Set the cache expiration value to 300 seconds, leave the default timeout value for web services (WS) operations and enter the WSDL URL for the producer in the text field and press the "Refresh & Save" button:

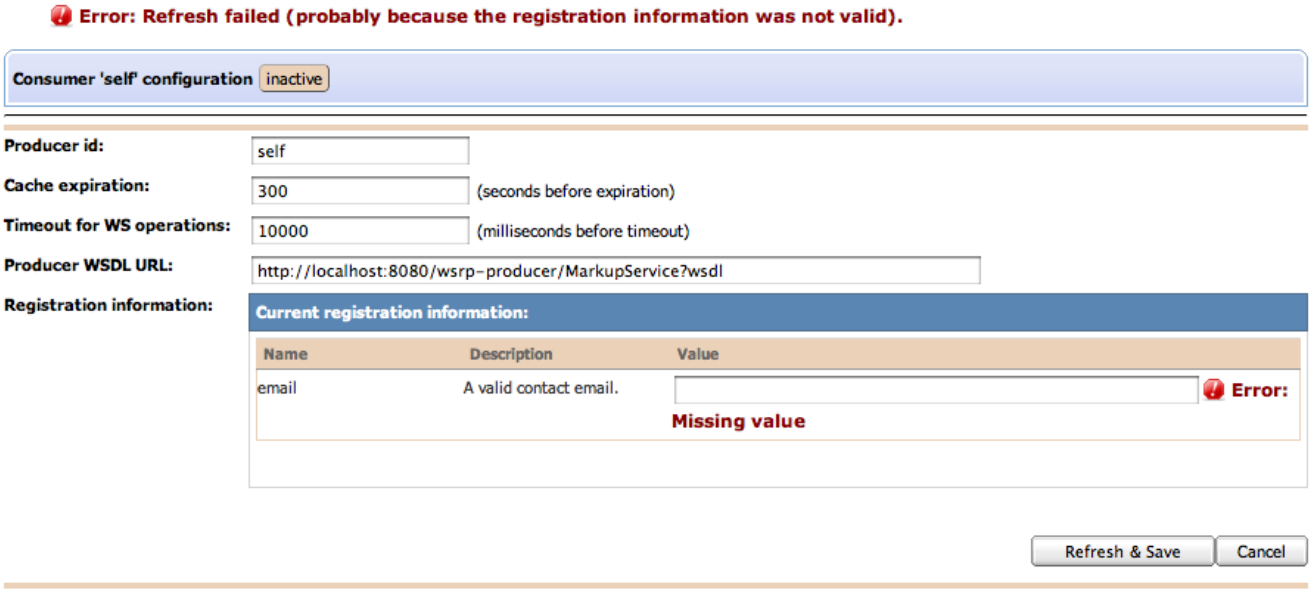


This will retrieve the service description associated with the Producer which WSRP interface is described by the WSDL file found at the URL you just entered. In this case, querying the service description will allow us to learn that the Producer requires registration but did not request any registration property:



The Consumer for the `oracle` Producer should now be available as a portlet provider and be ready to be used.

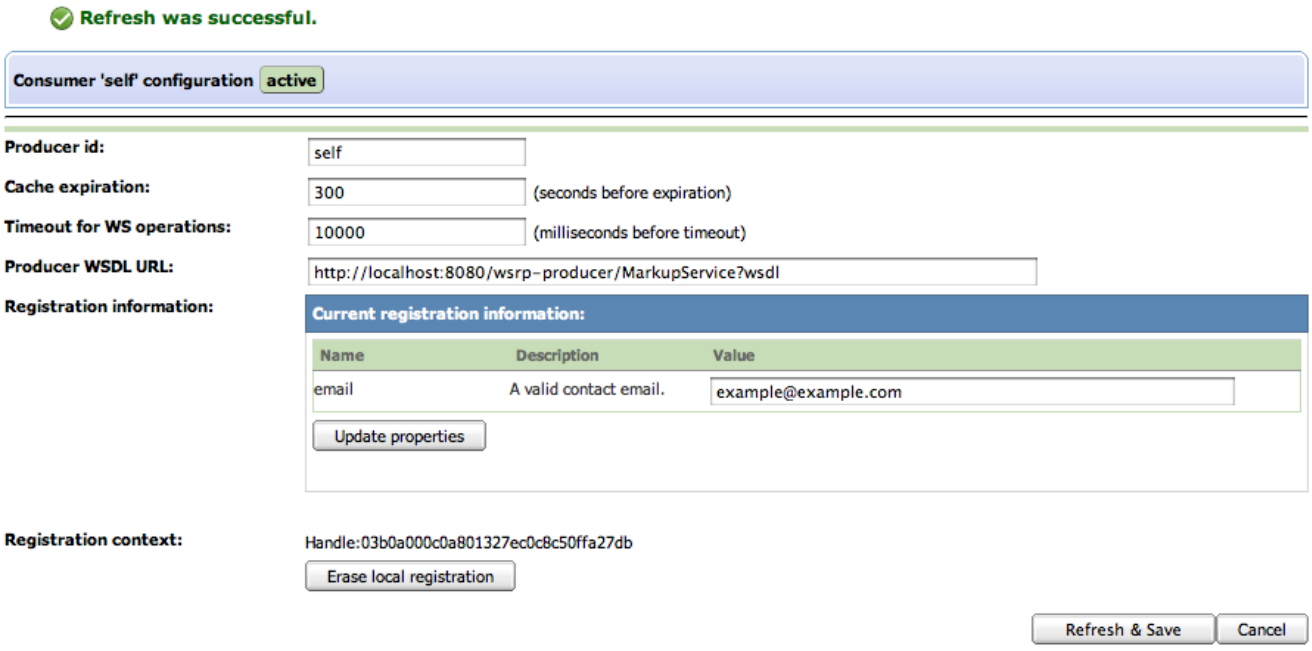
Now, assuming that the producer required a value for an `email` registration property, GateIn's WSRP consumer would have informed you that you were missing some information:



Note

At this point, there is no automated way to learn about which possible values (if any) are expected by the remote Producer. Sometimes, the possible values will be indicated in the registration property description but this is not always the case. Please refer to the specific Producer's documentation.

If you entered "`example@example.com`" as the value for the registration property and press "Save & Refresh" once more, you would have seen something similar to:



Using XML

While it is recommended that you use the WSRP Configuration portlet to configure Consumers, eXo Platform provides an alternative way to configure consumers by editing the .xml file located at `$GATEIN_HOME/lib/wsrp-consumer-$WSRP_VERSION.jar/conf/wsrp-consumers-config.xml`.

```
<?xml version='1.0' encoding='UTF-8' ?>
<deployments xmlns="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0 http://www.jboss.org/portal/xsd/gatein_wsrp_consumer_1_0.xsd">
  <deployment>
    <wsrp-producer id="self" expiration-cache="300" ws-timeout="30000">
      <endpoint-wsdl-url>http://localhost:8080/wsrp-producer/v2/MarkupService?wsdl</endpoint-wsdl-url>
       <registration-data>
         <property>
           <name>email</name>
           <lang>en</lang>
           <value>example@example.com</value>
         </property>
       </registration-data>
    </wsrp-producer>
  </deployment>
```

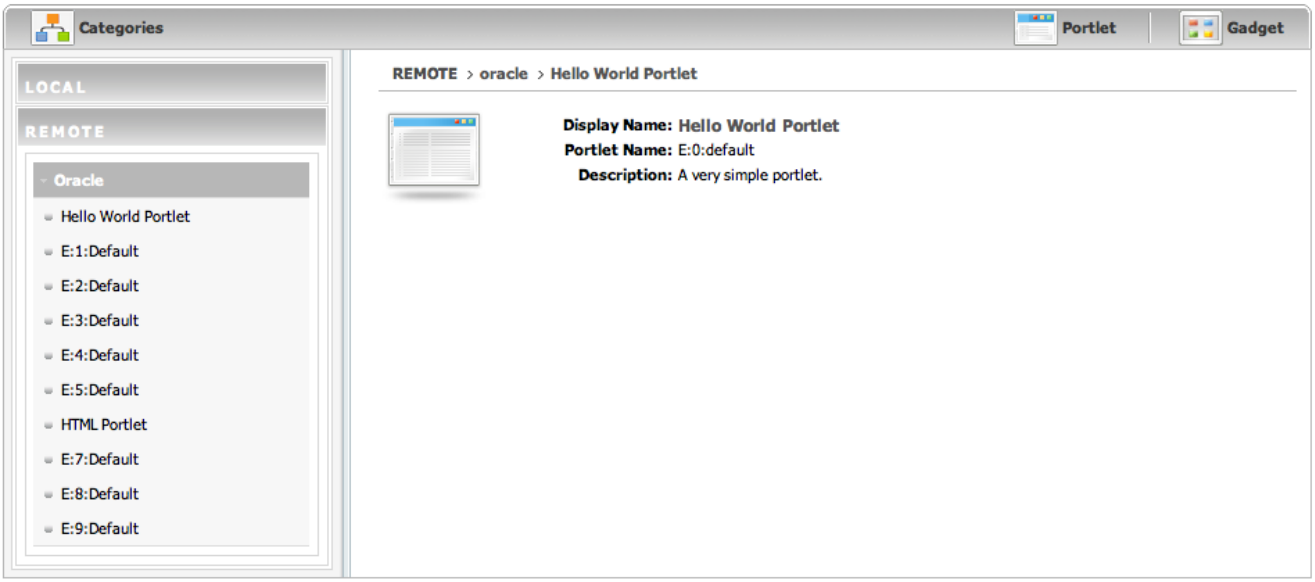
```
  <deployment>
    <wsrp-producer id="oracle" expiration-cache="300">
      <endpoint-wsdl-url>http://portalstandards.oracle.com/portletapp/portlets?WSDL</endpoint-
wsdl-url>
      <registration-data/>
    </wsrp-producer>
  </deployment>
</deployments>
```

The file as shown above specifies access to two producers: `self`, which consumes GateIn's own WSRP producer albeit in a version that assumes that the producer requires a value for an `email` registration property, and `oracle`, which consumes Oracle's public producer, both in configurations as shown in the walk-through above.

We will look at the details of the meaning of elements later on.

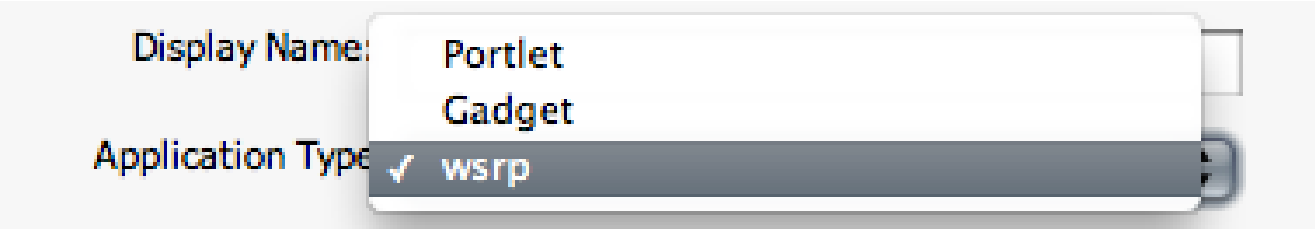
Adding remote portlets to categories

If we go back to the Application Registry and examine the available portlets by clicking the Portlet link, you will now be able to see the remote portlets after clicking the REMOTE tab in the left column:



These portlets are available to be used. For example, the regular portlets can be used in categories and added to pages. If you use the Import Applications functionality, they will also be automatically imported into categories based on the keywords they define.

More specifically, if you want to add the WSRP portlet to a category, you can access these portlets by selecting `wsrp` in the Application Type drop-down menu:



Configuring access to remote producers via XML

While it is recommended that you use the WSRP Configuration portlet to configure Consumers, eXo Platform provides an alternative way to configure consumers by editing the XML file located at `$GATEIN_HOME/lib/wsrp-consumer-$WSRP_VERSION.jar/conf/wsrp-consumers-config.xml`.

Note

An XML Schema defining which elements are available to configure Consumers via XML can be found in `$GATEIN_HOME/lib/wsrp-integration-api-$WSRP_VERSION.jar/xsd/gatein_wsrp_consumer_1_0.xsd`

Note

It is important to note how the XML consumers configuration file is processed. It is the first time the WSRP service starts and the associated information is then put under control of JCR (Java Content Repository). The subsequent launches of the WSRP service will use the JCR-stored information for all producers that are already known to GateIn. More specifically, the `wsrp-consumers-config.xml` file is scanned for producer identifiers. Any identifier that is already known will be by-passed and the JCR information associated with this remote producer will be used. The information defined at the XML level is only processed for producer definition for which no information is already present in JCR. Therefore, if you wish to delete a producer configuration, you need to delete the associated information in the database (this can be accomplished using the configuration portlet as shown in *the section called "Using the configuration portlet"*) *AND* remove the associated information in `wsrp-consumers-config.xml` (if such information exists) as the producer will be re-created the next time the WSRP is launched if that information is not removed.

Required configuration information

Now, look at which information needs to be provided to configure access to a remote producer.

First, you need to provide an identifier for the producer we are configuring so that you can refer to it afterwards. This is accomplished via the mandatory `id` attribute of the `<wsrp-producer>` element.

GateIn also needs to learn about the remote producer's end-points to be able to connect to the remote web services and perform WSRP invocations. This is accomplished by specifying the URL for the WSDL description for the remote WSRP service, using the `<endpoint-wsdl-url>` element.

Both the `id` attribute and `<endpoint-wsdl-url>` elements are required for a functional remote producer configuration.

Optional configuration

It is also possible to provide addtional configuration. In some cases, the additional configuration might be important to establish a proper connection to the remote producer.

One such optional configuration concerns caching. To prevent useless roundtrips between the local consumer and the remote producer, it is possible to cache some of the information sent by the producer (such as the list of offered portlets) for a given duration. The rate at which the information is refreshed is defined by the `expiration-cache` attribute of the `<wsrp-producer>` element which specifies the refreshing period in seconds. For example, providing a value of 120 for expiration-cache means that the producer information will not be refreshed for 2 minutes after it has been somehow accessed. If no value is provided, GateIn will always access the remote producer regardless of whether the remote information has changed or not. Since, in most instances, the information provided by the producer does not change often. It is recommended that you use this caching facility to minimize the bandwidth usage.

It is also possible to define a timeout after which the WS operations are considered as failed. This is helpful to avoid blocking the WSRP service, waiting forever on the service that does not answer. Use the `ws-timeout` attribute of the `<wsrp-producer>` element to specify how many milliseconds the WSRP service will wait for a response from the remote producer before timing out and giving up.

Additionally, some producers require consumers to register with them before authorizing them to access their offered portlets. If you know that information beforehand, you can provide the required registration information in the producer configuration so that the consumer can register with the remote producer when required.

Note

At this time, though, only simple String properties are supported and it is not possible to configure the complex registration data. This should, however, be sufficient for most cases.

Registration configuration is done via the `<registration-data>` element. Since GateIn can generate the mandatory information for you, if the remote producer does not require any registration properties, you only need to provide an empty `<registration-data>` element. Values for the registration properties required by the remote producer can be provided via the `<property>` elements. See the example below for more details. Additionally, you can override the default consumer name automatically provided by GateIn via the `<consumer-name>` element. If you select to provide a consumer name, please remember that this should uniquely identify your consumer.

Examples

Here is the configuration of the `selfv1` and `selfv2` consumers as found in `$GATEIN_HOME/lib/wsrp-consumer-$WSRP_VERSION.jar/conf/wsrp-consumers-config.xml` with a cache expiring every 500 seconds and with a 50 second timeout for web service operations.

Example 7.3.

```
<?xml version='1.0' encoding='UTF-8' ?>
<deployments xmlns="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0 http://
www.jboss.org/portal/xsd/gatein_wsrp_consumer_1_0.xsd">
  <deployment>
    <wsrp-producer id="selfv1" expiration-cache="500" ws-timeout="50000">
          <endpoint-wsdl-url>http://localhost:8080/wsrp-producer/MarkupService?wsdl</endpoint-
wsdl-url>
       <registration-data/>
    </wsrp-producer>
  </deployment>
  <deployment>
    <wsrp-producer id="selfv2" expiration-cache="500" ws-timeout="50000">
      <endpoint-wsdl-url>http://localhost:8080/wsrp-producer/v2/MarkupService?wsdl</endpoint-
wsdl-url>
       <registration-data/>
    </wsrp-producer>
  </deployment>
</deployments>
```

Here is an example of a WSRP descriptor with registration data and cache expiring every minute:

Example 7.4.

```
<?xml version='1.0' encoding='UTF-8' ?>
<deployments xmlns="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.gatein.org/xml/ns/gatein_wsrp_consumer_1_0 http://
www.jboss.org/portal/xsd/gatein_wsrp_consumer_1_0.xsd">
<deployments>
  <deployment>
```

```
    <wsrp-producer id="AnotherProducer" expiration-cache="60">
      <endpoint-wsdl-url>http://example.com/producer/producer?WSDL</endpoint-wsdl-url>
      <registration-data>
        <property>
          <name>property name</name>
          <lang>en</lang>
          <value>property value</value>
        </property>
      </registration-data>
    </wsrp-producer>
  </deployment>
</deployments>
```

Consumers maintenance

Modifying a currently held registration

Registration modification for service upgrade

Producers often offer several levels of service depending on consumers' subscription levels (for example). This is implemented at the WSRP level with the registration concept: producers can assert which level of service to provide to consumers based on the values of given registration properties.

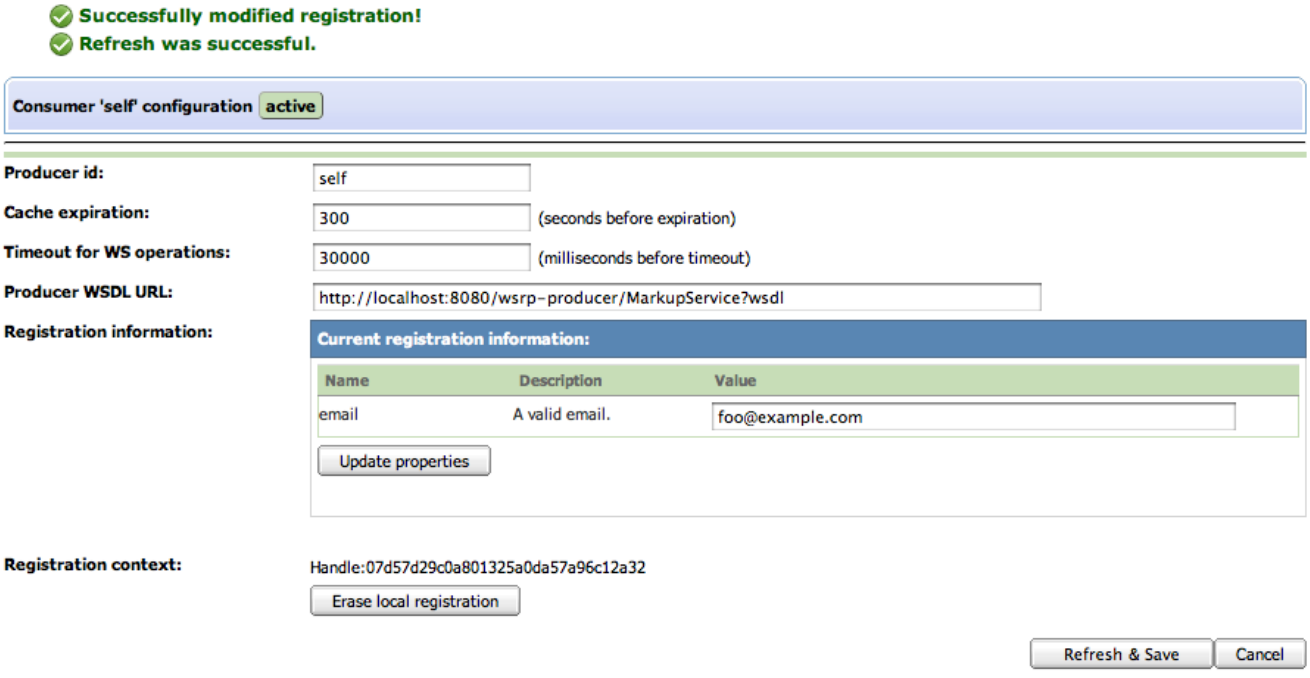
There might also be cases where you just want to update the registration information because it has changed. For example, the producer required you to provide a valid email and the previously email address is not valid anymore and needs to be updated.

It is therefore sometimes necessary to modify the registration that concretizes the service agreement between a consumer and a producer. Let's take the example of the producer requiring an email you configured in *the section called "Using the configuration portlet"*. If you recall, the producer was requiring registration and required a value to be provided for the `email` property.

Suppose that you want to update the email address that you provided to the remote producer. To do so, you need to tell the producer that our registration data have been modified. Let's see how to do this. In case that you want to configure access to the producer as previously described, please go to the configuration screen for the `self` producer and modify the value of `email` to `foo@example.com` instead of`example@example.com`:

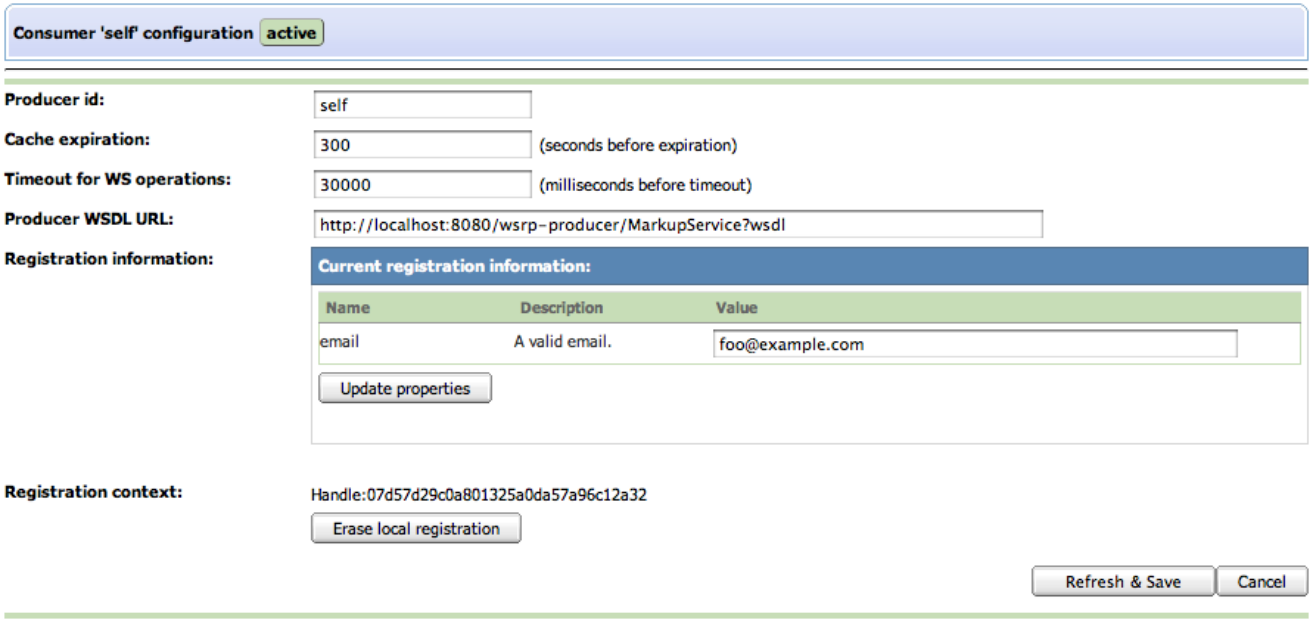
Now, click "Update properties" to save the change. A "Modify registration" button should now appear to let you send the new data to the remote producer:

Click this new button and, if everything went well and your updated registration has been accepted by the remote producer, you should see something similar to:

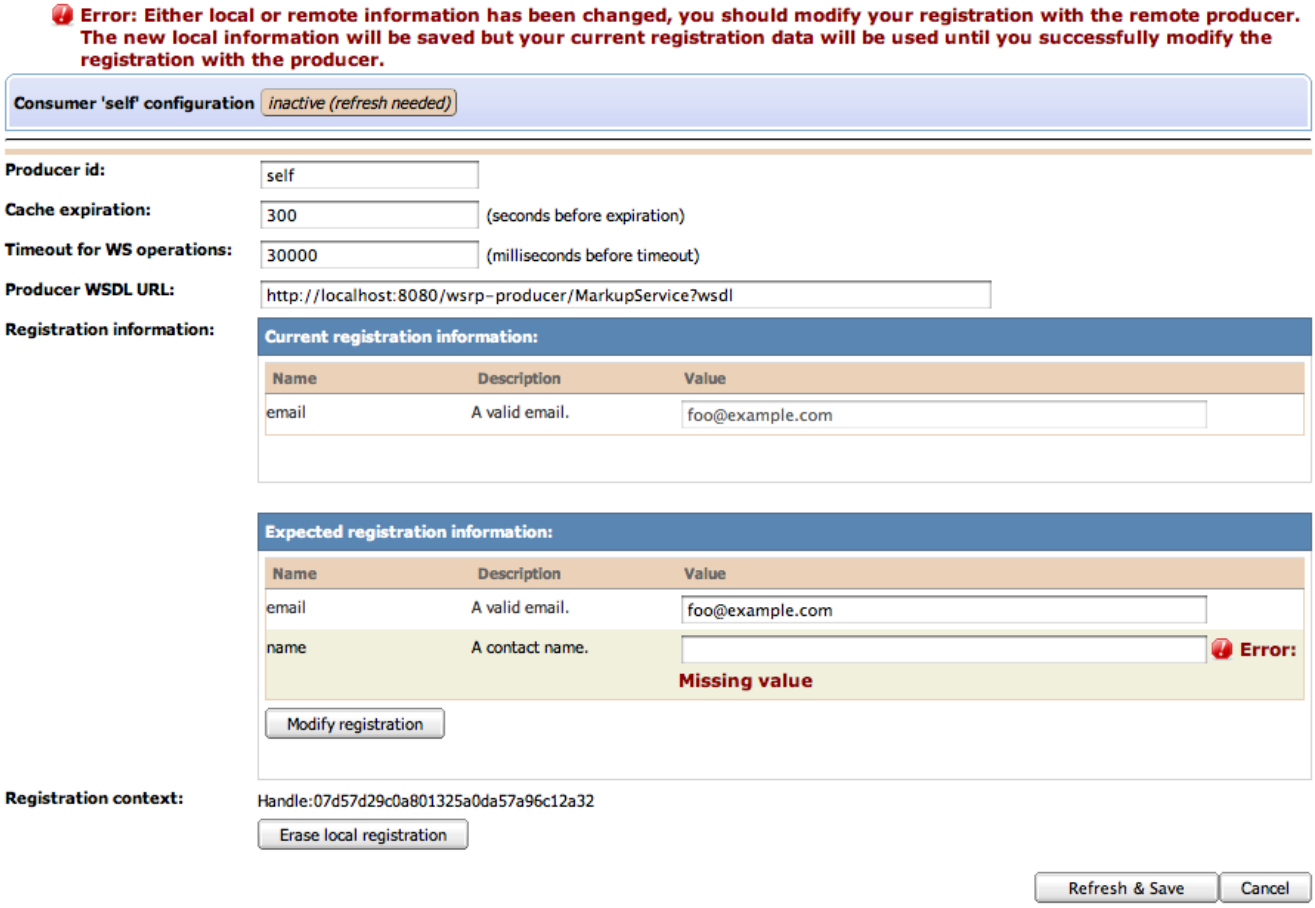


Registration modification on producer error

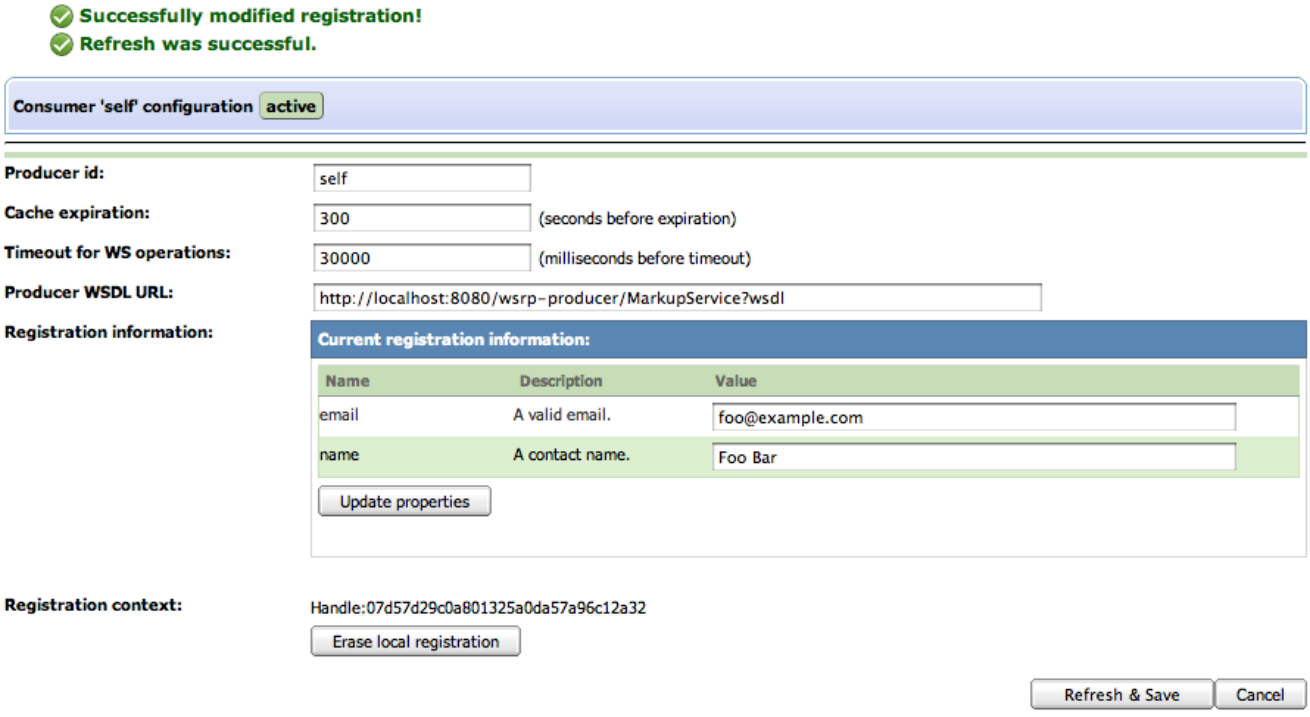
It can also happen that a producer administrator decided to change its requirement for registered consumers. In this case, invoking operations on the producer will be failed with an `OperationFailedFault`. GateIn will attempt to help you in this situation. Let's walk through an example using the `self` producer. Let's assume that registration is requiring a valid value for an `email` registration property (as we have seen so far). If you go to the configuration screen for this producer, you should see:



Now suppose that the administrator of the producer also requires a value to be provided for a `name` registration property. We will actually see how to perform this operation in GateIn when we examine how to configure GateIn's producer in *the section called "Configuring GateIn's WSRP Producer"*. Operations with this producer will now be failed. If you suspect that a registration modification is required, you should go to the configuration screen for this remote producer and refresh the information held by the consumer by pressing "Refresh & Save":



As you can see, the configuration screen now shows the currently held registration information and the expected information from the producer. Enter a value for the `name` property and then click "Modify registration". If all went on well and the producer accepted your new registration data, you should see something similar to:

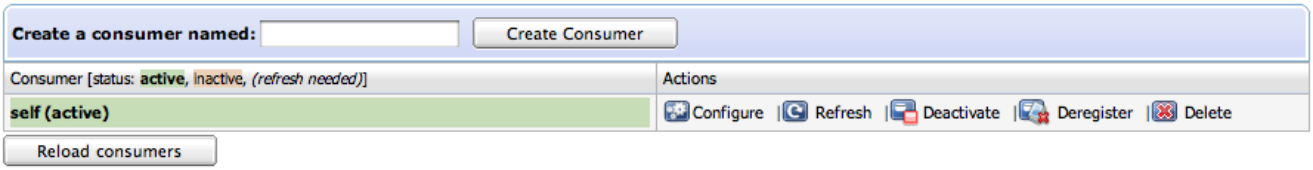


Note

As of WSRP 1, it is rather difficult to ascertain what caused an `OperationFailedFault` as it is the generic exception returned by producers if something did not quite happen as expected during a method invocation. This means that `OperationFailedFault` can be caused by several different reasons, one of them being a request to modify the registration data. Please take a look at the log files to see if you can gather more information as to what happened. WSRP 2 introduces an exception that is specific to a request to modify registrations, thus reducing the ambiguity that currently exists.

Consumer operations

Several operations are available from the consumer list view of the WSRP configuration portlet:



The available operations are:

- Configure: displays the consumer details and allows user to edit them.
- Refresh: forces the consumer to retrieve the service description from the remote producer to refresh the local information (offered portlets, registration information).

- Activate/Deactivate: activates/deactivates a consumer, governing whether it will be available to provide portlets and receive portlet invocations.
- Register/Deregister: registers/deregisters a consumer based on whether the registration is required and/or acquired
- Delete: destroys the consumer, after deregistering it if it was registered.
- Export: exports some or all of the consumer's portlets to be able to later import them in a different context
- Import: imports some or all of previously exported portlets

Note

Import/Export functionalities are only available to WSRP 2 consumers. Import functionality is only available if portlets had previously been exported.

Importing and exporting portlets

Import and export are new functionalities added in WSRP 2. Exporting a portlet allows a consumer to get an opaque representation of the portlet which can then be use by the corresponding import operation to reconstitute it. It is mostly used in migration scenarios during batch operations. Since GateIn does not currently support automated migration of portal data, the functionality that we provide as part of WSRP 2 is necessarily less complete than it could be with full portal support.

The import/export implementation in GateIn 3.2 allows users to export portlets from a given consumer. These portlets can then be used to replace existing content on pages. This is accomplished by assiging previously exported portlets to replace the content displayed by windows on the portal's pages. Let us walk through an example to make things clearer.

Clicking the "Export" action for a given consumer will display the list of portlets currently made available by this specific consumer. An example of such a list is shown below:

Once portlets have been selected, they can be exported by clicking the "Export" button thus making them available for later import:

You can re-import the portlets directly by pressing the "Use for import" button or, on the Consumers list page, using the "Import" action for a given consumer. Let's assume that you used that second option and that you currently have several available sets of previously exported portlets to import from. After clicking the action link, you should see a screen similar to the one below:

As you can see this screen presents the list of available exports with available operations for each.

- View: displays the export details as previously seen when the export was first performed
- Delete: deletes the selected export, asking you for confirmation first

- Use for import: selects the export to import portlets from

Once you have selected an export to import from, you will see a screen similar to the one below:

The screen displays the list of available exported portlets for the previously selected export. You can select which portlet you want to import by checking the checkbox next to its name. Next, you need to select the content of which window the imported portlet will replace. This process is done in three steps. Let's assume in this example that you have the following page called `page1` and containing two windows called `NetUnity WSRP 2 Interop - Cache Markup (remote)` and `/samples-remotecontroller-portlet.RemoteControl (remote)` as shown below:

In this example, we want to replace the content of the `/samples-remotecontroller-portlet.RemoteControl (remote)` by the content of the `/ajaxPortlet.JSFAJAXPortlet` portlet that we previously exported. To do so, we will check the checkbox next to the `/ajaxPortlet.JSFAJAXPortlet` portlet name to indicate that we want to import its data and then select the `page1` in the list of available pages. The screen will then refresh to display the list of available windows on that page, similar to the one seen below:

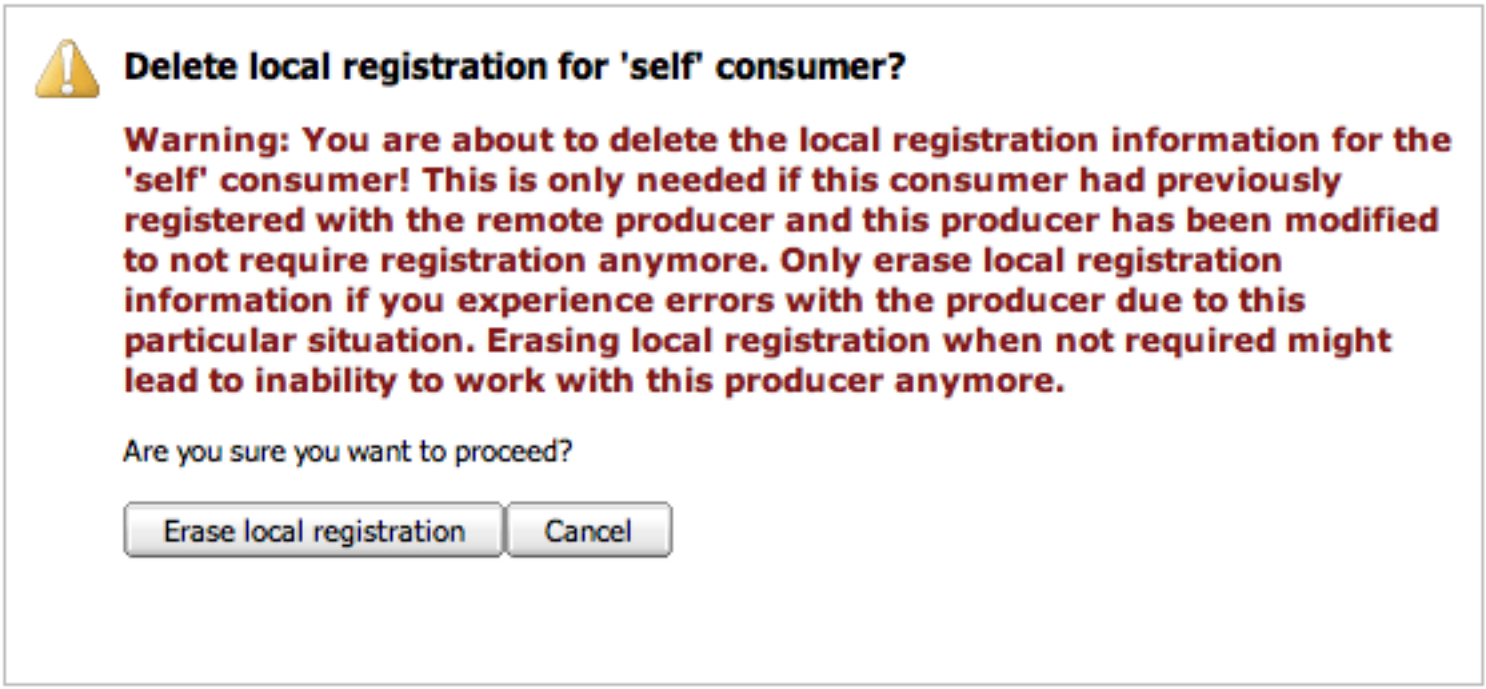
Note that, at this point, we still need to select the window which content we want to replace before being able to complete the import operation. Let's select the `/samples-remotecontroller-portlet.RemoteControl (remote)` window, at which point the "Import" button will become enabled, indicating that we now have all the necessary data to perform the import. If all goes well, pressing that button should result in a screen similar to the one below:

If you now take a look at the `page1` page, you should now see that the content `/samples-remotecontroller-portlet.RemoteControl (remote)` window has been replaced by the content of the `/ajaxPortlet.JSFAJAXPortlet` imported portlet and the window renamed appropriately:

Erasing local registration data

There are rare cases where it might be required to erase the local information without being able to deregister first. This is the case when a consumer is registered with a producer that has been modified by its administrator to not require the registration anymore. If that ever was to happen (most likely, it will not), you can erase the local registration information from the consumer so that it can resume interacting with the remote producer. To do so, click the "Erase local registration" button next to the registration context information on the consumer configuration screen:

Warning: This operation is dangerous as it can result in inability to interact with the remote producer if invoked when not required. A warning screen will be displayed to give you a chance to change your mind:



Configuring GateIn's WSRP Producer

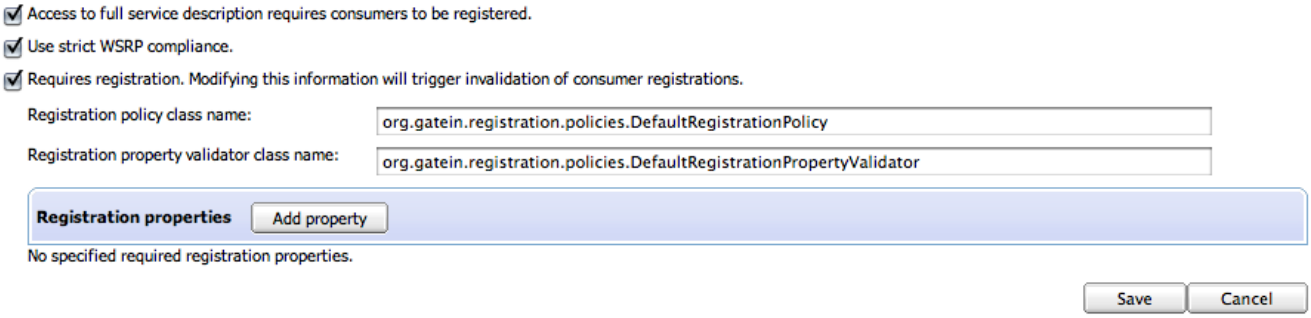
Overview

You can configure the behavior of Portal's WSRP Producer by using the WSRP administration interface, which is the preferred way, or by editing the `$GATEIN_HOME/wsrp-producer.war/WEB-INF/conf/producer/config.xml` file. Several aspects can be modified with respects to whether the registration is required for consumers to access the Producer's services. An XML Schema for the configuration format is available at `$GATEIN_HOME/lib/wsrp-integration-api-$WSRP_VERSION.jar/xsd/gatein_wsrp_producer_1_0.xsd` .

Default configuration

The default producer configuration requires consumers to register with it before providing access to its services but does not require any specific registration properties (apart from what is mandated by the WSRP standard). It does, however, require consumers to be registered before sending them a full service description. This means that our WSRP producer will not provide the list of offered portlets and other capabilities to unregistered consumers. The producer also uses the default `RegistrationPolicy` paired with the default `RegistrationPropertyValidator`. You need to look into property validators in greater detail later in*the section called "Registration configuration"*. This allows you to customize how Portal's WSRP Producer decides whether a given registration property is valid or not.

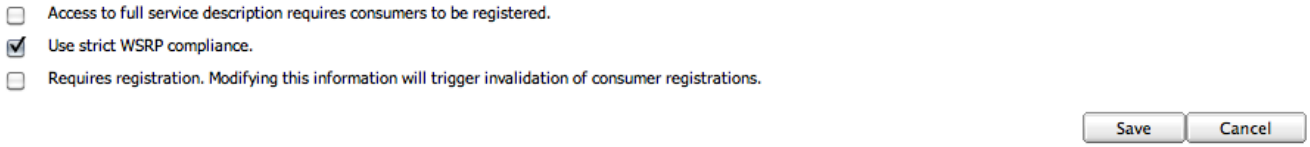
GateIn provides a web interface to configure the producer's behavior. You can access it by clicking the "Producer Configuration" tab of the "WSRP" page of the "admin" portal. Here's what you should see with the default configuration:



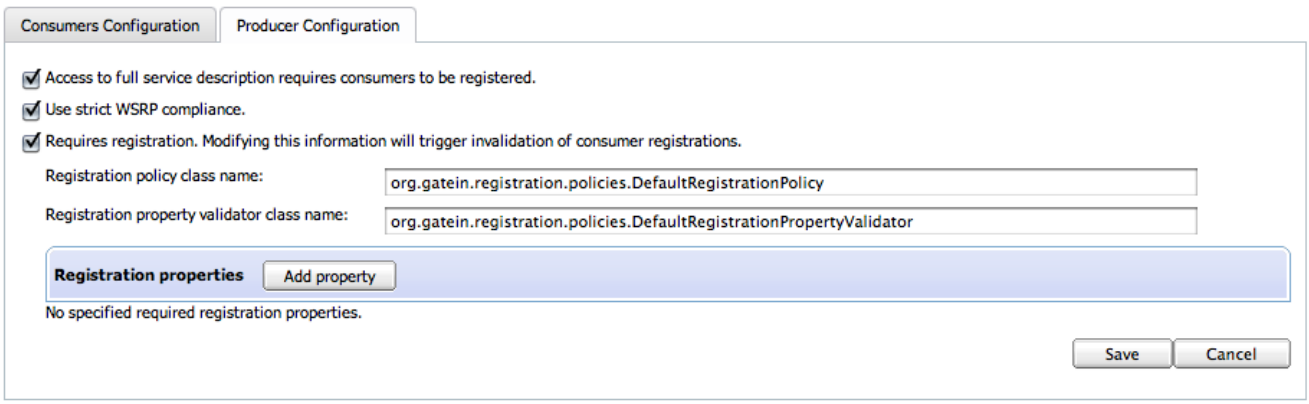
As expected, you can specify whether or not the producer will send the full service description to unregistered consumers, and, if it requires registration, which `RegistrationPolicy` to use (and, if needed, which `RegistrationPropertyValidator`), along with required registration property description for which consumers must provide acceptable values to register successfully.

Registration configuration

To require consumers to register with the Portal's producer before interacting with it, you need to configure the Portal's behavior with respect to registration. Registration is optional, as registration properties. The producer can require registration without passing any registration properties in case of the default configuration. Let's configure our producer starting with a blank state:

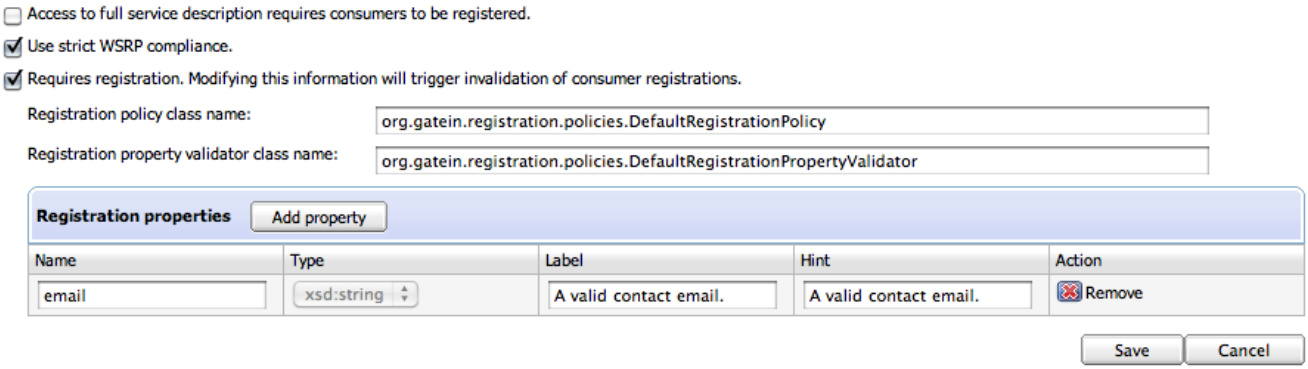


eXo Platform will allow unregistered consumers to see the list of offered portlets, so the first checkbox ("Access to full service description requires consumers to be registered") is unchecked. You will, however, specify that consumers will need to be registered to be able to interact with our producer. Check the second checkbox ("Requires registration. Modifying this information will trigger invalidation of consumer registrations."). The screen should now refresh and display:



You can specify the fully-qualified name for your `RegistrationPolicy` and `RegistrationPropertyValidator` there. We will keep the default value. See *the section*

called "Customization of Registration handling behavior" for more details. Let's add, however, a registration property called `email`. Click "Add property" and enter the appropriate information in the fields, providing a description for the registration property that can be used by consumers to figure out its purpose:



Press "Save" to record your modifications.

Note

At this time, only String (xsd:string) properties are supported. If your application requires more complex properties, please let us know.

Note

If consumers are already registered with the producer, modifying the configuration of required registration information will trigger the invalidation of held registrations, requiring consumers to modify their registration before being able to access the producer again. We saw the consumer side of that process in *the section called "Registration modification on producer error"*.

Customization of Registration handling behavior

Registration handling behavior can be customized by users to suit their Producer needs. This is accomplished by providing an implementation of the `RegistrationPolicy` interface. This interface defines methods that are called by Portal's Registration service so that decisions can be made appropriately. A default registration policy that provides basic behavior is provided and should be enough for most user needs.

While the default registration policy provides the default behavior for most registration-related aspects, there is still one aspect that requires configuration: whether a given value for a registration property is acceptable by the WSRP Producer. This is accomplished by plugging a `RegistrationPropertyValidator` in the default registration policy. This allows you to define their own validation mechanism.

Please refer to the Javadoc™ for `org.jboss.portal.registration.RegistrationPolicy` and `org.jboss.portal.Registration.policies.RegistrationPropertyValidator` for more details on what is expected of each method.

Defining a registration policy is required for the producer to be correctly configured. This is accomplished by specifying the qualified class name of the registration policy. Since we anticipate that most users will use the default registration policy, it is possible to provide the class name of your custom property validator instead of customizing the default registration policy behavior. Note that property validators are only used by the default policy.

Note

Since the policy or the validator are defined via their class name and dynamically loaded, it is important that you make sure that the identified class is available to the application server. One way to accomplish that is to deploy your policy implementation as the JAR file in your AS instance deploy directory. Note also that, since both policies and validators are dynamically instantiated, they must provide a default, no-argument constructor.

WSRP validation mode

The lack of conformance kit and the wording of the WSRP specification leaves room for differing interpretations, resulting in interoperability issues. It is therefore possible to encounter issues when using consumers from different vendors. We have experienced such issues and have introduced a way to relax the validation that our WSRP producer performs on the data provided by consumers to help with interoperability by accepting data that would normally be invalid. Note that we only relax our validation algorithm on aspects of the specification that are deemed harmless, such as invalid language codes.

By default, the WSRP producer is configured in the strict mode. If you experience issues with a given consumer, you might want to try to relax the validation mode. This is accomplished by unchecking the "Use strict WSRP compliance." checkbox on the Producer configuration screen.

Advanced Development

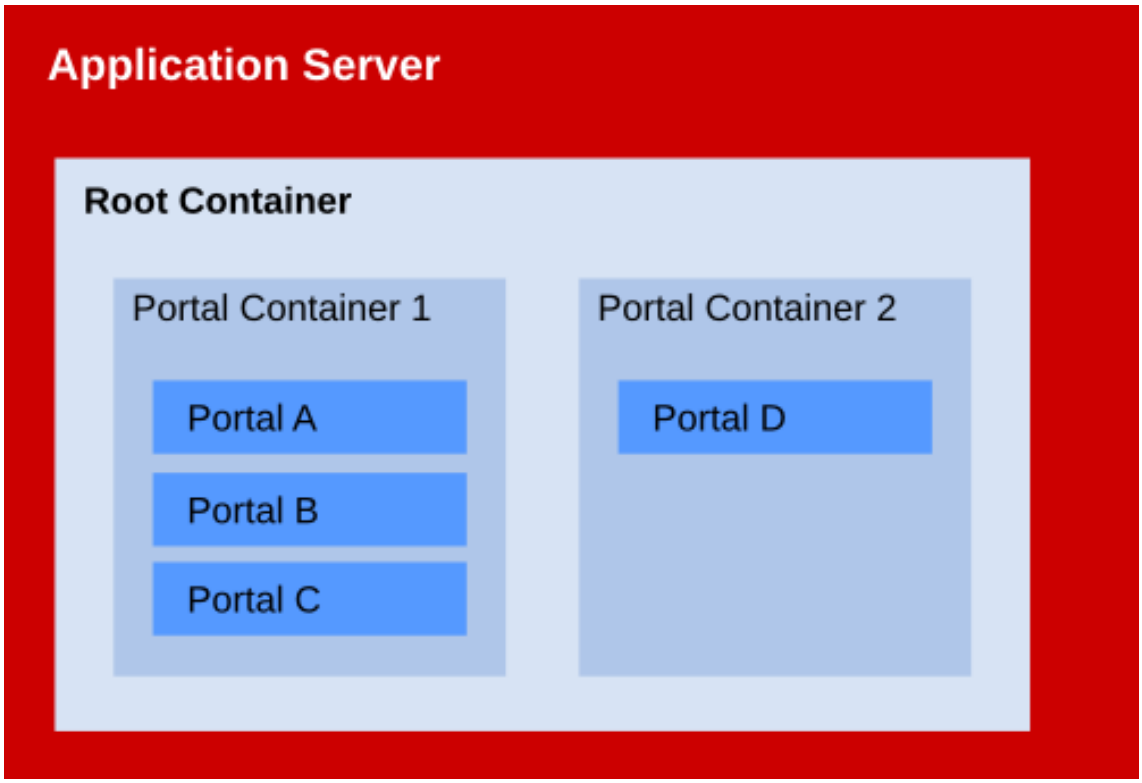
Foundations

GateIn Kernel

GateIn 3.2 is built as a set of services on top of the dependency injection kernel. The kernel provides configuration, lifecycle handling, component scopes, and some core services.

Service components exist in two scopes. The first scope is represented by **RootContainer** - it contains services that exist independently of any portal, and can be accessed by all portals.

The second scope is portal-private in the form of **PortalContainer**. Each portal lives in an instance of PortalContainer. This scope contains services that are common for a set of portals, and services which should not be shared by all portals.



Whenever a specific service is looked up through PortalContainer, and the service is not available, the lookup is delegated further up to RootContainer. We can therefore have the default instance of a certain component in RootContainer, and portal specific instances in some or all PortalContainers which override the default instance.

Whenever your portal application has to be integrated more closely with GateIn services, the way to do it is by looking up these services through PortalContainer. Be careful though - only officially documented services should be accessed this way, and used according to documentation, as most of the services are an implementation detail of GateIn, and subject to change without notice.

Configuring services

GateIn Kernel uses the dependency injection to create services based on the **configuration.xml** configuration files. The location of the configuration files determines if services are placed into the RootContainer scope, or into the PortalContainer scope. All configuration.xml files located at **conf/configuration.xml** in the classpath (any directory, or any jar in the classpath) will have their services configured at the RootContainer scope. All configuration.xml files located at **conf/portal/configuration.xml** in the classpath will have their services configured at the PortalContainer scope. Additionally, **portal extensions** can contain configuration in **WEB-INF/conf/configuration.xml**, and will also have their services configured at the PortalContainer scope.

Note

Portal extensions are described later.

Configuration syntax

Components

A service component is defined in the **configuration.xml file** by using **<component>** element.

There is only one required information when defining a service - the service implementation class, specified using **<type>**

Every component has a **<key>** that identifies it. If not explicitly set, a key defaults to the value of <type>. If the key is loaded as a class, the Class or String object will be used as a key.

The usual approach is to specify an interface as a key.

Example 8.1. Example of service component configuration:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd
                        http://www.exoplaform.org/xml/ns/kernel_1_0.xsd"
    xmlns="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd">
  <component>
    <key>org.exoplatform.services.database.HibernateService</key>
    <type>org.exoplatform.services.database.impl.HibernateServiceImpl</type>

    ...

  </component>
</configuration>
```

External Plugins

GateIn Kernel supports non-component objects that can be configured, instantiated, and injected into registered components, using method calls. The mechanism is called 'plugins', and allows portal extensions to add additional configurations to core services.

The external plugin is defined by using the **<external-component-plugins>** wrapper element which contains one or more **<component-plugin>** definitions. <external-component-plugins> uses **<target-component>** to specify a target service component that will receive injected objects.

Every <component-plugin> defines an implementation type, and a method on the target component to use for the injection (**<set-method>**).

A plugin implementation class has to implement the **org.exoplatform.container.component.ComponentPlugin** interface.

In the following example, **PortalContainerDefinitionPlugin** implements ComponentPlugin:

Example 8.2. PortalContainerDefinitionPlugin

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd
                        http://www.exoplaform.org/xml/ns/kernel_1_0.xsd"
    xmlns="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd">

  <external-component-plugins>
          <target-component>org.exoplatform.container.definition.PortalContainerConfig</target-component>
    <component-plugin>
      <!-- The name of the plugin -->
      <name>Add PortalContainer Definitions</name>

      <!-- The name of the method to call on the PortalContainerConfig
          in order to register the PortalContainerDefinitions -->
      <set-method>registerPlugin</set-method>

      <!-- The fully qualified name of the PortalContainerDefinitionPlugin -->
      <type>org.exoplatform.container.definition.PortalContainerDefinitionPlugin</type>

      ...
```

```
    </component-plugin>
  </external-component-plugins>
</configuration>
```

Includes, and special URLs

It is possible to break the **configuration.xml** file into many smaller files, that are then included into a 'master' configuration file. The included files are complete configuration.xml documents, not fragments of text.

The following is an example configuration.xml which 'outsources' its content into several files:

```
<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd
                  http://www.exoplaform.org/xml/ns/kernel_1_0.xsd"
    xmlns="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd">

  <import>war:/conf/sample-ext/jcr/jcr-configuration.xml</import>
  <import>war:/conf/sample-ext/portal/portal-configuration.xml</import>

</configuration>
```

The special URL is used to refer to another configuration file. The URL schema **'war:'** means that the path following is resolved that is related to the current PortalContainer's servlet context resource path, starting at **WEB-INF** as the root.

Note

The current PortalContainer is really a newly created PortalContainer, as war: URLs only make sense for PortalContainer scoped configuration.

Also, thanks to the extension mechanism, the servlet context used for resource loading is a **unified servlet context** (as explaned in a later section).

To include the resolved path related to the current classpath (context classloader), use the **'jar:'** URL schema.

Special variables

The configuration files may contain a **special variable** reference *${container.name.suffix}*. This variable resolves to the name of the current portal container, prefixed by underscore (_). This

facilitates reuse of configuration files in cases where portal specific unique names need to be assigned to some resources (For example, JNDI names, Database / DataSource names, JCR repository names).

This variable is only defined when there is a current PortalContainer available - only for PortalContainer scoped services.

A good example for this is **HibernateService**:

Example 8.3. HibernateService using variables

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd
                      http://www.exoplaform.org/xml/ns/kernel_1_0.xsd"
  xmlns="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd">

  <component>
    <key>org.exoplatform.services.database.HibernateService</key>
    <jmx-name>database:type=HibernateService</jmx-name>
    <type>org.exoplatform.services.database.impl.HibernateServiceImpl</type>
    <init-params>
      <properties-param>
        <name>hibernate.properties</name>
        <description>Default Hibernate Service</description>
        <property name="hibernate.show_sql" value="false" />
        <property name="hibernate.cglib.use_reflection_optimizer" value="true" />
        <property name="hibernate.connection.url"
                  value="jdbc:hsqldb:file:../temp/data/exodb${container.name.suffix}" />
        <property name="hibernate.connection.driver_class" value="org.hsqldb.jdbcDriver" />
        <property name="hibernate.connection.autocommit" value="true" />
        <property name="hibernate.connection.username" value="sa" />
        <property name="hibernate.connection.password" value="" />
        <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect" />
        <property name="hibernate.c3p0.min_size" value="5" />
        <property name="hibernate.c3p0.max_size" value="20" />
        <property name="hibernate.c3p0.timeout" value="1800" />
        <property name="hibernate.c3p0.max_statements" value="50" />
      </properties-param>
    </init-params>
  </component>
</configuration>
```

InitParams configuration object

`InitParams` are the configuration object that is essentially a map of key-value pairs, where key is always a `String`, and value can be any type that can be described using the kernel configuration.xml.

Service components that form GateIn 3.2 insfrastructure use the `InitParams` object to configure themselves. A component can have one instance of InitParams injected at most. If the service component's constructor takes InitParams as any of the parameters, it will automatically be injected at the component instantiation time. The xml configuration for a service component that expects the InitParams object must include the <init-params> element (even if the empty one).

To learn about how the kernel xml configuration syntax looks for creating `InitParams` instances, see the following example.

Example 8.4. InitParams - properties-param

```
<component>
  <key>org.exoplatform.services.naming.InitialContextInitializer</key>
  <type>org.exoplatform.services.naming.InitialContextInitializer</type>
  <init-params>
    <properties-param>
      <name>default-properties</name>
      <description>Default initial context properties</description>
      <property name="java.naming.factory.initial"
            value="org.exoplatform.services.naming.SimpleContextFactory" />
    </properties-param>
  </init-params>
</component>
```

The InitParams object description begins with the <init-params> element. It can have zero or more children elements, each of which is one of <value-param>, <values-param>, <properties-param>, or <object-param>. Each of these child elements takes a <name> that serves as a map entry key, and an optional <description>. It also takes a type-specific value specification.

For <properties-param>, the value specification is in the form of one or more <property> elements, each of which specifies two strings - a property name, and a property value. Each <properties-params> defines one `java.util.Properties` instance. Also, see *Example 8.3, "HibernateService using variables"* for an example.

Example 8.5. InitParams - value-param

```
<component>
  <key>org.exoplatform.services.transaction.TransactionService</key>
  <type>org.exoplatform.services.transaction.impl.jotm.TransactionServiceJotmImpl</type>
  <init-params>
    <value-param>
      <name>timeout</name>
      <value>5</value>
    </value-param>
  </init-params>
</component>
```

For <value-param>, the value specification is in the form of <value> element, which defines one `String` instance.

Example 8.6. InitParams - values-param

```
<component>
 <key>org.exoplatform.services.resources.ResourceBundleService</key>
 <type>org.exoplatform.services.resources.impl.SimpleResourceBundleService</type>
  <init-params>
    <values-param>
      <name>classpath.resources</name>
     <description>The resources  that start with the following package name should be load from
file system</description>
      <value>locale.portlet</value>
    </values-param>

    <values-param>
      <name>init.resources</name>
      <description>Store the following resources into the db for  the first launch </description>
      <value>locale.test.resources.test</value>
    </values-param>

    <values-param>
      <name>portal.resource.names</name>
      <description>The properties files of  the portal ,  those file will be merged
        into one ResourceBundle properties </description>
```

```
      <value>local.portal.portal</value>
      <value>local.portal.custom</value>
    </values-param>
  </init-params>
</component>
```

For <values-param>, the value specification is in the form of one or more <value> elements, each of which represents one `String` instance, where all the `String` values are then collected into a `java.util.List` instance.

Example 8.7. InitParams - object-param

```
<component>
  <key>org.exoplatform.services.cache.CacheService</key>
  <jmx-name>cache:type=CacheService</jmx-name>
  <type>org.exoplatform.services.cache.impl.CacheServiceImpl</type>
  <init-params>
    <object-param>
      <name>cache.config.default</name>
      <description>The default cache configuration</description>
      <object type="org.exoplatform.services.cache.ExoCacheConfig">
        <field name="name">
          <string>default</string>
        </field>
        <field name="maxSize">
          <int>300</int>
        </field>
        <field name="liveTime">
          <long>300</long>
        </field>
        <field name="distributed">
          <boolean>false</boolean>
        </field>
        <field name="implementation">
          <string>org.exoplatform.services.cache.concurrent.ConcurrentFIFOExoCache</string>
        </field>
      </object>
    </object-param>
  </init-params>
</component>
```

For <object-param>, the value specification comes in a form of the <object> element, which is used for the POJO style object specification (you specify an implementation class - <type>, and property values - <field>).

Also see *Example 8.8, "Portal container declaration example"* for an example of specifying a field of the `Collection` type.

The `InitParams` structure - the names and types of entries is specific for each service, as it is the code inside service components's class that decides what entry names to look up and what types it expects to find.

Configuring a portal container

A **portal container** is defined by several attributes.

First, there is a **portal container name**, which is always equal to the URL context to which the current portal is bound.

Second, there is a **REST context name**, which is used for REST access to portal application - every portal has exactly one (unique) REST context name.

Then, there is a **realm name** which is the name of security realm used for authentication when users log into the portal.

Finally, there is a list of **Dependencies** - other web applications, whose resources are visible to current portal (via extension mechanism described later), and are searched in the specified order.

Example 8.8. Portal container declaration example

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd
                      http://www.exoplaform.org/xml/ns/kernel_1_0.xsd"
  xmlns="http://www.exoplaform.org/xml/ns/kernel_1_0.xsd">

  <external-component-plugins>
    <!-- The full qualified name of the PortalContainerConfig -->
          <target-component>org.exoplatform.container.definition.PortalContainerConfig</target-
component>

    <component-plugin>
      <!-- The name of the plugin -->
```

```
<name>Add PortalContainer Definitions</name>

<!-- The name of the method to call on the PortalContainerConfig
     in order to register the PortalContainerDefinitions -->
<set-method>registerPlugin</set-method>

<!-- The full qualified name of the PortalContainerDefinitionPlugin -->
<type>org.exoplatform.container.definition.PortalContainerDefinitionPlugin</type>

<init-params>
  <object-param>
    <name>portal</name>
    <object type="org.exoplatform.container.definition.PortalContainerDefinition">
      <!-- The name of the portal container -->
      <field name="name"><string>portal</string></field>

      <!-- The name of the context name of the rest web application -->
      <field name="restContextName"><string>rest</string></field>

      <!-- The name of the realm -->
      <field name="realmName"><string>exo-domain</string></field>

      <!-- All the dependencies of the portal container ordered by loading priority -->
      <field name="dependencies">
        <collection type="java.util.ArrayList">
          <value>
            <string>eXoResources</string>
          </value>
          <value>
            <string>portal</string>
          </value>
          <value>
            <string>dashboard</string>
          </value>
          <value>
            <string>exoadmin</string>
          </value>
          <value>
            <string>eXoGadgets</string>
          </value>
          <value>
            <string>eXoGadgetServer</string>
          </value>
          <value>
```

```
                <string>rest</string>
              </value>
              <value>
                <string>web</string>
              </value>
              <value>
                <string>wsrp-producer</string>
              </value>
              <!-- The sample-ext has been added at the end of the dependency list
                   in order to have the highest priority -->
              <value>
                <string>sample-ext</string>
              </value>
            </collection>
          </field>
        </object>
      </object-param>
    </init-params>
  </component-plugin>
 </external-component-plugins>
</configuration>
```

Note

Dependencies are part of the extension mechanism.

Every **portal container** is represented by a **PortalContainer instance**, including:

- Associated **ExoContainerContext** containing information about the portal.
- **Unified servlet context** for web-archive-relative resource loading.
- **Unified classloader** for classpath based resource loading.
- Methods for retrieving services.

Unified servlet context, and **unified classloader** are part of the **extension mechanism** (explained in next section), and provide standard APIs (ServletContext, ClassLoader) with the specific resource loading behavior - visibility into the associated web application archives, configured with Dependencies property of PortalContainerDefinition. Resources from other web applications are queried in the order specified by Dependencies. The later entries in the list override the previous ones.

GateIn Extension Mechanism, and Portal Extensions

Extension mechanism is a functionality that makes it possible to override the portal resources in an almost plug-and-play fashion - just drop in a .war archive with the resources, and configure its position on the portal's classpath. This way any customizations of the portal do not have to involve unpacking and repacking the original portal .war archives. Instead, create your own .war archive with changed resources that override the resources in the original archive.

A web archive packaged in a way to be used through the extension mechanism is called **portal extension**.

There are two steps necessary to create a portal extension.

First, declare **PortalConfigOwner** servlet context listener in web.xml of your web application.

Example 8.9. Example of a portal extension called sample-ext:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE web-app PUBLIC -//Sun Microsystems, Inc.//DTD Web Application 2.3//EN
    http://java.sun.com/dtd/web-app_2_3.dtd>
<web-app>

  <display-name>sample-ext</display-name>

  <listener>
    <listener-class>org.exoplatform.container.web.PortalContainerConfigOwner</listener-class>
  </listener>

  ...

</web-app>
```

Then, add the servlet context name of this web application to a proper location in the list of Dependencies of the PortalContainerDefinition of all the portal containers that you want to access its resources.

After this step, your web archive will be on the portal's unified classpath, and unified servlet context resource path. The later in the Dependencies list your application is, the higher priority it has when resources are loaded by portal.

Note

See the 'Configuring a portal' section for example of PortalContainerDefinition, that has sample-ext at the end of its list of Dependencies.

Running Multiple Portals

It is possible to run several independent portal containers - each bound to a different URL context - within the same JVM instance. This kind of setup is very efficient from administration and resource consumption. The most elegant way to **reuse** configuration for different coexisting portals is by way of extension mechanism - by **inheriting** resources and configuration from existing web archives, and just **adding** extra resources to it, and **overriding** those that need to be changed by including modified copies.

In order for a portal application to correctly function when deployed in multiple portals, the application may have to dynamically query the information about the current portal container. The application should not make any assumptions about the name, and other information of the current portal, as there are now multiple different portals in play.

At any point during request processing, or lifecycle event processing, your application can retrieve this information through **org.exoplatform.container. ExoContainerContext**. Sometimes your application needs to make sure that the proper **PortalContainer** - the source of **ExoContainerContext** - is associated with the current call.

If you ship servlets or servlet filters as part of your portal application, and if you need to access the portal specific resources at any time during the processing of the servlet or filter request, you need to make sure the servlet/filter is associated with the current container.

The proper way to do that is to make your servlet extend **org.exoplatform.container.web. AbstractHttpServlet** class. This will not only properly initialize the current **PortalContainer** for you, but will also set the current thread's context classloader to one that looks for resources in the associated web applications in the order specified by **Dependencies** configuration (as explained in Extension mechanism section).

Similarly for filters, make sure your filter class extends **org.exoplatform.container.web. AbstractFilter**. Both **AbstractHttpServlet**, and **AbstractFilter** have the same method **getContainer()**, which returns the current **PortalContainer**. If your servlet handles the requests by implementing the **service()** method, you need to rename that method to match the following signature:

```
/**
 * Use this method instead of Servlet.service()
 */
protected void onService(ExoContainer container, HttpServletRequest req,
    HttpServletResponse res) throws ServletException, IOException;
```

Note

The reason is that AbstractHttpServlet implements service() to perform its interception, and you don't want to overwrite (by overriding) this functionality.

You may also need to access portal information within your **HttpSessionListener**. Again, make sure to extend the provided abstract class - **org.exoplatform.container.web.AbstractHttpSessionListener**. Also, modify your method signatures as follows:

```
/**
 * Use this method instead of HttpSessionListener.sessionCreated()
 */
protected void onSessionCreated(ExoContainer container, HttpSessionEvent event);

/**
 * Use this method instead of HttpSessionListener.sessionDestroyed()
 */
protected void onSessionDestroyed(ExoContainer container, HttpSessionEvent event);
```

There is another method you have to implement in this case:

```
/**
 * Method should return true if unified servlet context,
 * and unified classloader should be made available
 */
protected boolean requirePortalEnvironment();
```

If this method returns true, the current thread's context classloader is set up according to the **Dependencies** configuration, and availability of the associated web applications. If it returns false, the standard application separation rules are used for resource loading (effectively turning off the extension mechanism). This method exists on **AbstractHttpServlet** and **AbstractFilter** as well, where there is a default implementation that automatically returns true, when it detects there is a current PortalContainer present. Otherwises, it returns false.

The followings explain how to properly perform the **ServletContextListener** based initialization, when you need to access the current PortalContainer.

GateIn has no direct control over the deployment of application archives (.war, .ear files) - it is the application server performing the deployment. For the **extension mechanism** to work

properly, the applications, associated with the portal via the **Dependencies** configuration, have to be deployed before the portal, that depends on them, is initialized. On the other hand, these applications may require an already initialized PortalContainer to properly initialize themselves - we have a recursive dependency problem. To resolve this problem, a mechanism of **initialization tasks**, and **task queues**, was put in place. Web applications that depend on the current PortalContainer for their initialization have to avoid performing their initialization directly in some ServletContextListener executed during their deployment (before any PortalContainer was initialized). Instead, a web application should package its initialization logic into an init task of the appropriate type, and only use ServletContextListener to insert the init task instance into the proper init tasks queue.

An example of this is the Gadgets application which registers the Google gadgets with the current PortalContainer:

```
public class GadgetRegister implements ServletContextListener
{
  public void contextInitialized(ServletContextEvent event)
  {
    // Create a new post-init task
    final PortalContainerPostInitTask task = new PortalContainerPostInitTask() {

      public void execute(ServletContext context, PortalContainer portalContainer)
      {
        try
        {
          SourceStorage sourceStorage =
          (SourceStorage) portalContainer.getComponentInstanceOfType(SourceStorage.class);
          ...
        }
        catch (RuntimeException e)
        {
          throw e;
        }
        catch (Exception e)
        {
          throw new RuntimeException("Initialization failed: ", e);
        }
      }
    };

    // Add post-init task for execution on all the portal containers
    // that depend on the given ServletContext according to
    // PortalContainerDefinitions (via Dependencies configuration)
```

```
    PortalContainer.addInitTask(event.getServletContext(), task);
  }
}
```

The above example uses **PortalContainerPostInitTask**, which gets executed after the portal container has been initialized. In some cases, you may want to execute the initialization after portal container was instantiated, but before it was initialized - use **PortalContainerPreInitTask** in that case. Or, you may want to execute initialization after all the post-init tasks have been executed - use **PortalContainerPostCreateTask** in that case.

Also, you may need to pay attention to **LoginModules**. If you use custom LoginModules which require the current ExoContainer, make sure they extend **org.exoplatform.services.security.jaas.AbstractLoginModule** for the proper initialization. AbstractLoginModule also takes care of the basic configuration - it recognizes two initialization options - **portalContainerName**, and **realmName** whose values you can access via protected fields of the same name.

eXo Platform 3.0 - Content Functions

Preface

Get Started with eXo Content

eXo Content is a fully integrated content management solution inside eXo Platform. Basically, in eXo Platform, the extension mechanism is used to add content features. To have more information about the extension mechanism, refer to the *GateIn reference guide* [http://docs.jboss.com/gatein/portal/3.1.0-FINAL/reference-guide/en-US/html/index.html].

This integrated solution provides users with a comprehensive platform about integrating applications, managing and publishing content - all from a single, familiar console.

When starting a new project, you can see eXo Content as a Java developer platform.

This document will give you guidelines related to building stable applications using eXo Content from the inside. The document consists of the following main contents:

- ***Application*** describes portlet applications included in eXo Content.
- ***Configuration*** provides you the comprehensive knowledge about the configuration with a large range of configuration parameters declared in eXo Content.
- ***Developer References*** describes about extension, public APIs, Java APIs, FAQs and related others in eXo Content.

Package

All package actions in eXo Content are started from the *delivery* folder, so you should go to this folder first.

```
$ cd ecms/delivery
```

- Package WCM standalone version - Tomcat bundle

```
$ cd wcm/assembly
$ mvn clean install
```

- Package WCM with workflow enabled - Tomcat bundle

```
$ cd wkf-wcm/assembly
$ mvn clean install
```

- Make WCM EAR packages to run with jBoss

1. Make WCM extension EAR

```
$ cd packaging/wcm/ear
$ mvn clean install
```

2. Make WCM demo sites EAR

```
$ cd packaging/ecmdemo/ear
$ mvn clean install
```

3. Make workflow EAR

```
$ cd packaging/workflow/ear
$ mvn clean install
```

Applications

This chapter provides you an comprehensive view about portlet applications included in eXo Content. These portlet applications are packaged as Web application archives (WARs).

Also, you can specify the package of each portlet and its available preferences that allow you to extend the configuration choices for standard preferences defined in *portlet.xml*.

Portlets

Content Detail

The **Content Detail** portlet allows users to view the detail of a specific content.

This is an example of the **Content Detail** portlet used in eXo Content:

- **Packaging**: This portlet is packaged in the *presentation.war* file.
- **The portlet class name**: *org.exoplatform.wcm.webui.scv.UISingleContentViewerPortlet*
- **Available preferences**: When using this portlet, you can customize the following preferences:

Preference	Type	Value	Description
repository	`String`	`repository`	The repository where data are stored and maintained.
workspace	`String`	`collaboration`	The workspace where content is stored.
nodeIdentifier	`String`	`N/A`	The UUID or the path of content that you want to show.
ShowTitle	`Boolean`	`true`	Show the content title on the top of the portlet.
ShowDate	`Boolean`	`false`	Show the content date on the top of the portlet.
ShowOptionBar	`Boolean`	`false`	Show a bar with some actions (Print, Back).
ContextEnable	`Boolean`	`false`	Define if the portlet will use the parameter on

Preference	Type	Value	Description
			URL as the path to content to display or not.
ParameterName	`String`	`content-id`	Define which parameter will be used to get the content's path.
ShowVote	`Boolean`	`false`	Show the result of voting for the displayed content.
ShowComments	`Boolean`	`false`	Show the existing comments of this content (if any).
sharedCache	`Boolean`	`true`	Define if the portlet will cache the displayed contents.

- **Sample configuration**

```
<portlet-preferences>
  <preference>
    <name>repository</name>
    <value>repository</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>workspace</name>
    <value>collaboration</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>nodeIdentifier</name>
    <value>/myfolder/mycontent</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>ShowTitle</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <preference>
```

```
    <name>ShowDate</name>
    <value>false</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>ShowOptionBar</name>
    <value>false</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>ShowPrintAction</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>isQuickCreate</name>
    <value>false</value>
    <read-only>true</read-only>
  </preference>
  <preference>
    <name>ContextEnable</name>
    <value>false</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>ParameterName</name>
    <value>content-id</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>sharedCache</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
</portlet-preferences>
```

Note

In WCM 2.3.0, some preferences are no longer used, for example, the ShowPrintAction preference.

Content List

The **Content List** portlet shows a list of contents which already exist in the system.

This is an example of the **Content List** portlet used in eXo Content:

- **Packaging**: This porlet is packaged in the *presentation.war* file.
- **The portlet class name**: *org.exoplatform.wcm.webui.clv.UICLVPortlet*
- **Available preferences**: When using this portlet, you can customize the following preferences:

Preference	Type	Value	Description
mode	`String`	`AutoViewerMode`	The mode for displaying content of the portlet: all contents in a specific folder or all specific contents in the portlet.
folderPath	`String`		The path to the folder whose contents are displayed by this portlet.
orderBy	`String`	`publication:liveDate`	The property by which all the contents in the portlet are sorted.
orderType	`String`	`DESC`	The type of the content sort method: ascending or descending.
header	`String`		The header of the portlet which is displayed at the top of the portlet.
automaticDetection	`Boolean`	`true`	This value indicates whether the header of the portlet is selected to be the title of the folder given in the **folderPath** parameter (true value) or the value given in the **header** paramter above.
formViewTemplatePath	`String`	`/exo:ecm/views/templates/content-`	The path to the template used to

Preference	Type	Value	Description
		`list-viewer/list/UIContentListPresentationDefault.gtmpl`	display the contents in this portlet.
paginatorTemplatePath	`String`	`/exo:ecm/views/templates/content-list-viewer/paginators/UIPaginatorDefault.gtmpl`	The path to the paginator used to display the contents in this portlet.
itemsPerPage	`Integer`	10	The number of contents displayed in every "page" of the portlet.
showThumbnailsView	`Boolean`	`true`	This value indicates whether the content image in this portlet is shown or not.
showTitle	`Boolean`	`true`	This value indicates whether the content title in this portlet is shown or not.
showHeader	`Boolean`	`true`	This value indicates whether the content header in this portlet is shown or not.
showRefreshButton	`Boolean`	`false`	This value indicates whether the **Refresh** button is shown in this portlet or not.
showDateCreated	`Boolean`	`true`	This value indicates whether the content created date in this portlet is shown or not.
showReadmore	`Boolean`	`true`	This value indicates whether the **Read more** button is shown in every content of the portlet or not. After clicking this button, the user can read the whole text of the content.

Preference	Type	Value	Description
showSummary	`Boolean`	`true`	This value indicates whether the content summary in this portlet is shown or not.
showLink	`Boolean`	`true`	If this value is **true**, the header of every content is also the link to view this content fully. If the value is **false**, the header is considered as a simple text.
showRssLink	`Boolean`	`true`	Show the RSS link of this portlet.
basePath	`String`	`detail`	Show the page in which the full content is displayed when the user clicks to the **Read more** button.
contextualFolder	`String`	`contextualDisable`	Enable/disable the contextual mode of the portlet. If enabled, the portlet can take the folder path indicated in the URL to display contents.
showScvWith	`String`	`content-id`	The parameter name which shows the folder path in URL when the **Read more** button is clicked.
showClvBy	`String`	`folder-id`	The parameter name which shows the folder path in URL.
sharedCache	`Boolean`	`true`	Define if the portlet will cache the displayed contents.

- **Sample Configuration**

```
<portlet-preferences>
  <preference>
    <name>mode</name>
    <value>AutoViewerMode</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>folderPath</name>
    <value/>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>orderBy</name>
    <value>publication:liveDate</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>orderType</name>
    <value>DESC</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>header</name>
    <value/>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>automaticDetection</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>formViewTemplatePath</name>
                                        <value>/exo:ecm/views/templates/content-list-viewer/list/
UIContentListPresentationDefault.gtmpl</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>paginatorTemplatePath</name>
      <value>/exo:ecm/views/templates/content-list-viewer/paginators/UIPaginatorDefault.gtmpl</
value>
    <read-only>false</read-only>
```

```
  </preference>
  <preference>
    <name>itemsPerPage</name>
    <value>10</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showThumbnailsView</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showTitle</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showHeader</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showRefreshButton</name>
    <value>false</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showDateCreated</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showReadmore</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showSummary</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showLink</name>
    <value>true</value>
```

```
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showRssLink</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>basePath</name>
    <value>detail</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>contextualFolder</name>
    <value>contextualDisable</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showScvWith</name>
    <value>content-id</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showClvBy</name>
    <value>folder-id</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>sharedCache</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
</portlet-preferences>
```

Search

The **Search** portlet allows users to do a search with any string. In eXo Content, there are three types of search: quick search, advanced search and search with saved queries.

The users can find this porlet in the front page. This is an example of the **Search** portlet used in eXo Content:

- **Packaging**: This portlet is packaged in the *searches.war* file.
- **The portlet class name**: *org.exoplatform.wcm.webui.search.UIWCMSearchPortlet*
- **Available preferences**: When using this portlet, you can customize the following preferences:

Preference	Type	Value	Description
repository	`String`	`repository`	The place where data are stored and maintained.
workspace	`String`	`collaboration`	The workspace where the content is stored.
searchFormTemplatePath	`String`	`/exo:ecm/views/templates/WCM Advance Search/search-form/UIDefaultSearchForm.gtmpl`	The path to the search form template.
searchResultTemplatePath	`String`	`/exo:ecm/views/templates/WCM Advance Search/search-result/UIDefaultSearchResult.gtmpl`	The path to the search result template.
searchPaginatorTemplatePath	`String`	`/exo:ecm/views/templates/WCM Advance Search/search-paginator/UIDefaultSearchPaginator.gtmpl`	The path to the search paginator template.
searchPageLayoutTemplatePath	`String`	`/exo:ecm/views/templates/WCM Advance Search/search-page-layout/UISearchPageLayoutDefault.gtmpl`	The path to the search page template.
itemsPerPage	`Integer`	5	The number of items for each page.
showQuickEditButton	`Boolean`	`true`	Show or hide the quick edit icon.
basePath	`String`	`parameterizedviewer`	The page which is used to display the search result.

- **Sample configuration**

```
<portlet-preferences>
  <preference>
    <name>repository</name>
    <value>repository</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>workspace</name>
    <value>collaboration</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>searchFormTemplatePath</name>
                    <value>/exo:ecm/views/templates/WCM Advance Search/search-form/
UIDefaultSearchForm.gtmpl</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>searchResultTemplatePath</name>
                  <value>/exo:ecm/views/templates/WCM Advance Search/search-result/
UIDefaultSearchResult.gtmpl</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>searchPaginatorTemplatePath</name>
                <value>/exo:ecm/views/templates/WCM Advance Search/search-paginator/
UIDefaultSearchPaginator.gtmpl</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>searchPageLayoutTemplatePath</name>
              <value>/exo:ecm/views/templates/WCM Advance Search/search-page-layout/
UISearchPageLayoutDefault.gtmpl</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>itemsPerPage</name>
    <value>5</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showQuickEditButton</name>
```

```
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>basePath</name>
    <value>parameterizedviewer</value>
    <read-only>false</read-only>
  </preference>
</portlet-preferences>
```

Content Explorer

The **Content Explorer** portlet is used to manage all documents in different drives. With this portlet, users can do many different actions depending on their roles, such as adding/deleting a category and a document, showing/hiding a node, managing publication, and more.

This is an example of the **Content Explorer** portlet used in eXo Content:

- **Packaging**: The porlet is packaged in the *ecmexplorer.war* file.
- **The portlet class name**: *org.exoplatform.ecm.webui.component.explorer.UIJCRExplorerPortlet*
- **Available preferences**: When using this portlet, you can customize the following preferences:

Preference	Type	Value	Description
repository	`String`	`repository`	The repository name which is used in an instance of Content Explorer.
workspace	`String`	N/A	Not in use. The workspace name was included in the Drive.
path	`String`	N/A	The path of the node. This preference will be used when the selected usecase is **Parameterize**.
drive	`String`	N/A	Not in use. Replaced by the driveName preference.
views	`String`	N/A	

Preference	Type	Value	Description
			Not in use. The views will be displayed basing on the Drive which the user has the access permission.
allowCreateFolders	`String`	N/A	Allow creating a folder by type. When you do not specify the value, the default value will be nt:unstructured, nt:folder.
categoryMandatoryWhenFileUpload	`Boolean`	false	Force a user to add a category when uploading or creating a document.
uploadFileSizeLimitMB	`Float`	150	The maximum size of a file that is uploaded to the system (MB).
usecase	`String`	`selection`	The behavior to access Content Explorer. By default, the "selection" option is configured. Besides "selection", there are four other ways to configure the Content Explorer: **Jailed**, **Personal**, **Social**, **Parameterize**.
driveName	`String`	`Private`	The name of drive which the user wants to access.
trashHomeNodePath	`String`	`/Trash`	The location to store the deleted nodes.
trashRepository	`String`	`repository`	The name of the repository where stores the deleted nodes.
trashWorkspace	`String`	`collaboration`	The name of the workspace where

Preference	Type	Value	Description
			stores the deleted nodes.
editInNewWindow	`Boolean`	`false`	Allow editing documents with or without a window popup.
showTopBar	`Boolean`	`true`	Allow showing the Top bar or not.
showActionBar	`Boolean`	`true`	Allow showing the Action bar or not.
showSideBar	`Boolean`	`true`	Allow showing the Side bar or not.
showFilterBar	`Boolean`	`true`	Allow showing the Filter bar or not.

- **Sample Configuration**

```
<portlet-preferences>
  <preference>
    <name>repository</name>
    <value>repository</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>workspace</name>
    <value/>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>path</name>
    <value/>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>drive</name>
    <value/>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>views</name>
```

```
  <value/>
  <read-only>false</read-only>
</preference>
<preference>
  <name>allowCreateFolders</name>
  <value/>
  <read-only>false</read-only>
</preference>
<preference>
  <name>categoryMandatoryWhenFileUpload</name>
  <value>false</value>
  <read-only>false</read-only>
</preference>
<preference>
  <name>uploadFileSizeLimitMB</name>
  <value>150</value>
  <read-only>false</read-only>
</preference>
<preference>
  <name>usecase</name>
  <value>selection</value>
  <read-only>false</read-only>
</preference>
<preference>
  <name>driveName</name>
  <value>Private</value>
  <read-only>false</read-only>
</preference>
<preference>
  <name>trashHomeNodePath</name>
  <value>/Trash</value>
  <read-only>false</read-only>
</preference>
<preference>
  <name>trashRepository</name>
  <value>repository</value>
  <read-only>false</read-only>
</preference>
<preference>
  <name>trashWorkspace</name>
  <value>collaboration</value>
  <read-only>false</read-only>
</preference>
<preference>
```

```
    <name>editInNewWindow</name>
    <value>false</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showTopBar</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showActionBar</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showSideBar</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showFilterBar</name>
    <value>true</value>
    <read-only>false</read-only>
  </preference>
</portlet-preferences>
```

Administration

The **Administration** portlet is used to manage the main ECM functions, including categories and tags, content presentation, types of content and advanced configuration.

This is an example of the **Administration** portlet used in eXo Content:

- **Packaging**: This portlet is packaged in the *ecmadmin.war* file.
- **The portlet class name**: *org.exoplatform.ecm.webui.component.admin.UIECMAdminPortlet*
- **Available preferences**: When using this portlet, you can customize the following preference:

Preference	Type	Value	Description
repository	`String`	`Repository`	The name of the current repository.

- **Sample Configuration**

```
<portlet-preferences>
  <preference>
    <name>repository</name>
    <value>repository</value>
    <read-only>false</read-only>
  </preference>
</portlet-preferences>
```

Fast Content Creator

The **Fast Content Creator** portlet consists of two modes: **Standard Content Creator** and **Basic Content Creator**. This portlet allows users to quickly create contents without accessing the Content Explorer portlet.

This is an example of the **Fast Content Creator** portlet used in eXo Content:

By default, this porlet is applied for the Contact Us portlet in eXo Content.

- **Packaging**: This portlet is packaged in the *formgenerator.war* file.
- **The portlet class name**: *org.exoplatform.wcm.webui.fastcontentcreator.UIFCCPortlet*
- **Available preferences**: When using this portlet, you can customize the following preferences:

Preference	Type	Value	Description
mode	`String`	`basic`	The default mode of the **Fast Content Creator** portlet.
repository	`String`	`repository`	The name of the current repository.
workspace	`String`	`collaboration`	The workspace where the content is stored.
path	`String`	`/Groups/platform/users/Documents`	The destination path where the content is stored.
type	`String`	`exo:article`	The node type of document which is

Preference	Type	Value	Description
			shown on the dialog form.
saveButton	`String`	`Save`	The custom button: **Save**.
saveMessage	`String`	`This node has been saved successfully`	The custom message when the user clicks the **Save** button.
isRedirect	`Boolean`	`false`	Specify whether redirecting to another page or not.
redirectPath	`String`	*http://www.google.com.vn*	The path to which the page will redirect.
isActionNeeded	`Boolean`	`true`	Specify whether an action is needed to save to the configuration or not.

- **Sample Configuration**

```
<portlet-preferences>
  <preference>
    <name>mode</name>
    <value>basic</value>
    <read-only>true</read-only>
  </preference>
  <preference>
    <name>repository</name>
    <value>repository</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>workspace</name>
    <value>collaboration</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>path</name>
    <value>/Groups/platform/users/Documents</value>
    <read-only>false</read-only>
  </preference>
```

```
  <preference>
    <name>type</name>
    <value>exo:article</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>saveButton</name>
    <value>Save</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>saveMessage</name>
    <value>This node has been saved successfully</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>isRedirect</name>
    <value>false</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>redirectPath</name>
    <value>http://www.google.com.vn</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>isActionNeeded</name>
    <value>true</value>
    <read-only>true</read-only>
  </preference>
</portlet-preferences>
```

Form Builder

The **Form Builder** portlet allows users to create node types and document templates for node types.

This is an example of the **Form Builder** portlet used in eXo Content:

- **Packaging**: This porlet is packaged in the *formgenerator.war* file.
- **The portlet class name**: *org.exoplatform.wcm.webui.formgenerator.UIFormGeneratorPortlet*

- **Available preferences**: When using this portlet, you can customize the following preference:

Preference	Type	Value	Description
repository	`String`	`repository`	The current repository name which is always "repository".

- **Sample Configuration**

```
<portlet-preferences>
  <preference>
    <name>repository</name>
    <value>repository</value>
    <read-only>false</read-only>
  </preference>
</portlet-preferences>
```

Authoring

The **Authoring** portlet allows users to manage contents in draft and ones which need to be approved or published.

This is an example of the **Authoring** portlet used in eXo Content:

- **Packaging**: This porlet is packaged in the *authoring-apps.war* file.

- **The portlet class name**: *org.exoplatform.wcm.webui.authoring.UIWCMDashboardPortlet*

- **Available preferences**: When using this portlet, you can customize the following preferences:

Preference	Type	Value	Description
repository	`String`	`Repository`	The name of the repository.
workspace	`String`	`Collaboration`	The name of the workspace.
drive	`String`	`Collaboration`	The name of the drive.

- **Sample Configuration**

```
<portlet-preferences>
  <preference>
    <name>repository</name>
    <value>repository</value>
    <read-only>true</read-only>
  </preference>
  <preference>
    <name>workspace</name>
    <value>collaboration</value>
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>drive</name>
    <value>collaboration</value>
    <read-only>false</read-only>
  </preference>
</portlet-preferences>
```

Newsletter

The **Newsletter** portlet is used to help users quickly get the updated newsletter from a site.

This is an example of the **Newsletter** portlet used in eXo Content:

- **Packaging**: This portlet is packaged in the *newsletter.war* file.
- **The portlet class name**: *org.exoplatform.wcm.webui.newsletter.manager.UINewsletterManagerPortlet*

SEO portlet

The **SEO** portlet allows users to manage SEO data of web content and web pages, so they can maximize their website position on search engines.

This is an example of the **SEO** portlet used in eXo Content:

- **Packaging**: This porlet is packaged in the *seo.war* file.
- **The portlet class name**: *org.exoplatform.wcm.webui.seo.UISEOToolbarPortlet*

Configuration

This chapter describes about configuration used in eXo Content. It consists of two main sections:

- *Component*
- *External Component Plugins*

Components

This section describes services which provide low-level functionality for UI components.

ActionServiceContainer

The *ActionServiceContainer* component is used to manage actions (adding, removing, or executing actions, and more) in the system. The configuration of this component is found in */core/core-configuration/src/main/webapp/WEB-INF/conf/wcm-core/core-services-configuration.xml*.

```
<component>
   <key>org.exoplatform.services.cms.actions.ActionServiceContainer</key>
   <type>org.exoplatform.services.cms.actions.impl.ActionServiceContainerImpl</type>
   <init-params>
      <value-param>
         <name>workspace</name>
         <value>system</value>
      </value-param>
      <value-param>
         <name>repository</name>
         <value>repository</value>
      </value-param>
   </init-params>
</component>
```

Details:

- **Value-param**:

Name	Type	Value	Description
workspace	`String`	`system`	The workspace name.
repository	`String`	`repository`	The repository name.

ApplicationTemplateManagerService

The *ApplicationTemplateManagerService* component is used to manage dynamic Groovy templates for WCM-based products. The configuration of this component is found in */core/core-configuration/src/main/webapp/WEB-INF/conf/wcm-core/core-services-configuration.xml*.

```
<component>
   <key>org.exoplatform.services.cms.views.ApplicationTemplateManagerService</key>
      <type>org.exoplatform.services.cms.views.impl.ApplicationTemplateManagerServiceImpl</
type>
   <init-params>
      <properties-param>
         <name>storedLocations</name>
         <property name="repository" value="system"/>
      </properties-param>
   </init-params>
</component>
```

Details:

- **Properties-param**:

Name	Property name	Type	Value	Description
storedLocations	`repository`	`String`	`system`	The repository name.

FragmentCacheService

The *FragmentCacheService* component is used to cache the response fragments which are sent to end-users.

The configuration of this component is found in *core/core-configuration/src/main/webapp/WEB-INF/conf/wcm-core/wcm-configuration.xml*.

```
<component>
   <key>org.exoplatform.services.portletcache.FragmentCacheService</key>
   <type>org.exoplatform.services.portletcache.FragmentCacheService</type>
   <init-params>
      <value-param>
         <name>cleanup-cache</name>
         <description>The cleanup cache period in seconds</description>
         <value>300</value>
```

```
      </value-param>
   </init-params>
</component>
```

Details

- **Value-param**:

Name	Type	Value	Description
cleanup-cache	`integer`	`300`	The time period over which cache items are expired.

JodConverterService

The *JodConverterServices* component is used to convert documents into different office formats. The configuration of this component is found in */core/core-configuration/src/main/webapp/WEB-INF/conf/wcm-core/core-services-configuration.xml*.

```
<component>
   <key>org.exoplatform.services.cms.jodconverter.JodConverterService</key>
   <type>org.exoplatform.services.cms.jodconverter.impl.JodConverterServiceImpl</type>
   <init-params>
      <value-param>
         <name>host</name>
         <value>127.0.0.1</value>
      </value-param>
      <value-param>
         <name>port</name>
         <value>8100</value>
      </value-param>
   </init-params>
</component>
```

Details:

- **Value-param**:

Name	Type	Value	Description
host	`The host IP`	`127.0.0.1`	The host from which a document is converted into

Name	Type	Value	Description
			different office formats.
port	`The port number`	`8100`	The port number is open to accept converting a document into different office formats.

LiveLinkManagerService

The *LiveLinkManagerService* component is used to check broken links, update links when the links are edited and extract links to return a list of all links. The configuration of this component is found in */core/core-configuration/src/main/webapp/WEB-INF/conf/wcm-core/wcm-configuration.xml*.

```
<component>
   <key>org.exoplatform.services.wcm.link.LiveLinkManagerService</key>
   <type>org.exoplatform.services.wcm.link.LiveLinkManagerServiceImpl</type>
   <init-params>
      <properties-param>
         <name>server.config</name>
         <description>server.address</description>
         <property name="scheme" value="http"/>
         <property name="hostName" value="localhost"/>
         <property name="port" value="8080"/>
      </properties-param>
   </init-params>
</component>
```

Details:

Properties-param	Property name	Type	Value	Description

server.config	`scheme`
	`hostName`
	`port`
		`http/https`
		`String`
		`The port number`
			`http`
			`localhost`
			`8080`
				All the property names are used together to configure the server. Here is an example about the server

Properties-param	Property name	Type	Value	Description
				configuration: http// :localhost:8080.

LockService

The *LockService* component is used to manage all locked nodes and allows unlocking the locked nodes in the system. It is also used to assign the Lock right to a user or a user group or a membership. The configuration of this component is found in */core/core-configuration/src/main/webapp/WEB-INF/conf/wcm-core/core-services-configuration.xml*.

```
<component>
  <key>org.exoplatform.services.cms.lock.LockService</key>
  <type>org.exoplatform.services.cms.lock.impl.LockServiceImpl</type>
</component>
```

NewsletterInitializationService

The *NewsletterInitializationService* component is used to initiate some data for the newsletter portlet. The configuration of this component is found in *packaging/ecmdemo/webapp/src/main/webapp/WEB-INF/conf/sample-portal/wcm/newsletter-configuration.xml*.

```
<component>
   <type>org.exoplatform.services.wcm.newsletter.NewsletterInitializationService</type>
   <init-params>
      <values-param>
         <name>portalNames</name>
         <value>classic</value>
         <value>acme</value>
      </values-param>
      <values-param>
         <name>administrators</name>
         <value>root</value>
         <value>john</value>
      </values-param>
      <object-param>
         <name>marketing</name>
         <description>marketing</description>
         <object type="org.exoplatform.services.wcm.newsletter.NewsletterCategoryConfig">
```

```
            <field name="name">
               <string>marketing</string>
            </field>
            <field name="title">
               <string>Marketing</string>
            </field>
            <field name="description">
               <string>You want to know where we are, where we go ?</string>
            </field>
            <field name="moderator">
               <string>*:/platform/web-contributors</string>
            </field>
         </object>
      </object-param>
      <object-param>
         <name>general</name>
         <description>general</description>
         <object type="org.exoplatform.services.wcm.newsletter.NewsletterCategoryConfig">
            <field name="name">
               <string>general</string>
            </field>
            <field name="title">
               <string>General</string>
            </field>
            <field name="description">
               <string>General information about us</string>
            </field>
            <field name="moderator">
               <string>*:/platform/web-contributors</string>
            </field>
         </object>
      </object-param>
      <object-param>
         <name>subscription2</name>
         <description>subscription2</description>
         <object type="org.exoplatform.services.wcm.newsletter.NewsletterSubscriptionConfig">
            <field name="name">
               <string>results</string>
            </field>
            <field name="title">
               <string>Results</string>
            </field>
            <field name="description">
               <string>Monthly newsletter about our results and forecasts</string>
```

```
            </field>
            <field name="categoryName">
               <string>general</string>
            </field>
            <field name="redactor">
               <string>*:/platform/web-contributors</string>
            </field>
         </object>
      </object-param>
      <object-param>
         <name>subscription1</name>
         <description>subscription1</description>
         <object type="org.exoplatform.services.wcm.newsletter.NewsletterSubscriptionConfig">
            <field name="name">
               <string>checklist</string>
            </field>
            <field name="title">
               <string>Check-List</string>
            </field>
            <field name="description">
               <string>Weekly newsletter with general topics</string>
            </field>
            <field name="categoryName">
               <string>general</string>
            </field>
            <field name="redactor">
               <string>*:/platform/web-contributors</string>
            </field>
         </object>
      </object-param>
      <object-param>
         <name>subscription3</name>
         <description>subscription3</description>
         <object type="org.exoplatform.services.wcm.newsletter.NewsletterSubscriptionConfig">
            <field name="name">
               <string>market</string>
            </field>
            <field name="title">
               <string>The market</string>
            </field>
            <field name="description">
               <string>What's on the market today ?</string>
            </field>
            <field name="categoryName">
```

```
            <string>marketing</string>
          </field>
          <field name="redactor">
            <string>*:/platform/web-contributors</string>
          </field>
        </object>
      </object-param>
      <object-param>
        <name>user1@gmail.com</name>
        <description>user1@gmail.com</description>
        <object type="org.exoplatform.services.wcm.newsletter.config.NewsletterUserConfig">
          <field name="mail">
            <string>user1@gmail.com</string>
          </field>
        </object>
      </object-param>
      <object-param>
        <name>user2@gmail.com</name>
        <description>user2@gmail.com</description>
        <object type="org.exoplatform.services.wcm.newsletter.config.NewsletterUserConfig">
          <field name="mail">
            <string>user2@gmail.com</string>
          </field>
        </object>
      </object-param>
    </init-params>
</component>
```

Details:

- **Value-param**

Name	Type	Value	Description
portalNames	`String`	`classic, acme`	The portal names.
administrators	`String`	`root`	The administrator who manages the newsletter portlet.

- **Object-param**:
  - **object type**: `org.exoplatform.services.wcm.newsletter.NewsletterCategoryConfig`

Field	Type	Description
name	`String`	The name of categories or subscriptions.

Field	Type	Description
title	`String`	The title of categories or subscriptions.
description	`String`	The description of categories or subscriptions.
moderator	`String`	The users or groups that can moderate the newsletter portlet.
categoryName	`String`	The categories name.
redactor	`String`	The users or groups that are newsletter redactor.
mail	`String`	The email that user uses to subscribe.

NewsletterManagerService

The *NewsletterManagerService* component is used to send newsletters to subscribers. The configuration of this component is found in *core/core-configuration/src/main/webapp/WEB-INF/conf/wcm-core/ext-newsletter-configuration.xml*.

```
<component>
   <type>org.exoplatform.services.wcm.newsletter.NewsletterManagerService</type>
   <init-params>
      <value-param>
         <name>repository</name>
         <value>repository</value>
      </value-param>
      <value-param>
         <name>workspace</name>
         <value>collaboration</value>
      </value-param>
   </init-params>
</component>
```

Details:

- **Value-param**:

Name	Type	Value	Description
repository	`String`	`repository`	The repository name.
workspace	`String`	`collaboration`	The workspace name.

SiteSearchService

The *SiteSearchService* component is used in the Search portlet that allows users to find all information matching with your given keyword.

It is configured in the *core/core-configuration/src/main/webapp/WEB-INF/conf/configuration.xml* file as follows:

```
<import>war:/conf/wcm-core/core-search-configuration.xml</import>
```

The component configuration maps the *SiteSearchService* component with its own implementation: *SiteSearchServiceImpl*.

```
<component>
  <key>org.exoplatform.services.wcm.search.SiteSearchService</key>
  <type>org.exoplatform.services.wcm.search.SiteSearchServiceImpl</type>
  <component-plugins>
    ...
  </component-plugins>
  <init-params>
    <value-param>
      <name>isEnabledFuzzySearch</name>
      <value>true</value>
    </value-param>
    <value-param>
      <name>fuzzySearchIndex</name>
      <value/>
    </value-param>
  </init-params>
</component>
```

Detail:

- **Value-param**:

Name	Type	Value	Description
isEnabledFuzzySearch	`Boolean`	`true`	Allow administrators to enable/disable the fuzzy search mechanism.

Name	Type	Value	Description
fuzzySearchIndex	N/A	N/A	Allow the approximate level between the input keyword and the found key results. In case of the invalid configuration, the default value is set to 0.8.

To have more information about the fuzzy search, please refer to *Fuzzy Search* [http://lucene.apache.org/java/2_4_1/queryparsersyntax.html#Fuzzy%20Searches].

SEOService

The *SEOService* component is used to help users manage SEO data of a page or a content, so their websites can achieve higher rankings on search engines. The configuration of this component is found in */packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/wcm-extension/wcm/seo-configuration.xml*.

```
<component>
  <key>org.exoplatform.services.seo.SEOService</key>
  <type>org.exoplatform.services.seo.impl.SEOServiceImpl</type>
  <init-params>
    <object-param>
      <name>seo.config</name>
      <object type="org.exoplatform.services.seo.SEOConfig">
        <field name="robotsindex">
          <collection type="java.util.ArrayList">
            <value>
              <string>INDEX</string>
            </value>
            <value>
              <string>NOINDEX</string>
            </value>
          </collection>
        </field>
        <field name="robotsfollow">
          <collection type="java.util.ArrayList">
            <value>
              <string>FOLLOW</string>
            </value>
            <value>
```

```
                    <string>NOFOLLOW</string>
                </value>
            </collection>
        </field>
        <field name="frequency">
            <collection type="java.util.ArrayList">
                <value>
                    <string>Always</string>
                </value>
                <value>
                    <string>Hourly</string>
                </value>
                <value>
                    <string>Daily</string>
                </value>
                <value>
                    <string>Weekly</string>
                </value>
                <value>
                    <string>Monthly</string>
                </value>
                <value>
                    <string>Yearly</string>
                </value>
                <value>
                    <string>Never</string>
                </value>
            </collection>
        </field>
    </object>
  </object-param>
 </init-params>
</component>
```

Details:

- **Object-param**:
  - **Object type**: `org.exoplatform.services.seo.SEOConfig`

Field	Type	Value	Description
robotsindex	`ArrayList`	`INDEX`	Allow search engines to index a particular page or not.

Field	Type	Value	Description
		`NOINDEX`	
robotsfollow	`ArrayList`	`FOLLOW` `NOFOLLOW`	Allow search engines to follow links from a particular page to find other pages or not.

frequency	`ArrayList`	`Always`
		`Hourly`
		`Daily`
		`Weekly`
		`Monthly`
		`Yearly`
		`Never`
			Define how often a particular page is updated.

QueryService

The *QueryService* component is used to manage many queries, including adding, removing or executing a query. The configuration of this component is found in */core/core-configuration/src/main/webapp/WEB-INF/conf/wcm-core/core-services-configuration.xml*.

```
<component>
   <key>org.exoplatform.services.cms.queries.QueryService</key>
   <type>org.exoplatform.services.cms.queries.impl.QueryServiceImpl</type>
   <init-params>
      <value-param>
         <name>workspace</name>
         <value>system</value>
      </value-param>
      <value-param>
         <name>relativePath</name>
         <value>Private/Queries</value>
      </value-param>
      <value-param>
         <name>group</name>
```

```
        <value>*:/admin</value>
      </value-param>
   </init-params>
</component>
```

Details:

- **Value-param**:

Name	Type	Value	Description
workspace	`String`	`system`	The workspace name.
relativePath	`Private/Queries`	`Private/Queries`	The path to the query location.
group	`String`	*:/admin	The group is allowed to access the query folder.

TaxonomyService

The *TaxonomyService* component is used to sort documents to ease searches when browsing documents online. It provides a multi-dimensional set of paths to find a document. In many cases, you can get your content by using different category paths. Therefore, after creating a document somewhere in the repository, it is possible to categorize it by adding several taxonomy references. By browsing the taxonomy tree, it will be possible to find the referencing article and display them as if they were children of the taxonomy nodes. Taxonomies are stored in the JCR itself and the JCR Reference functionality is used to provide that advanced WCM feature. The tree of taxonomies can be managed simply, such as copying/cutting/pasting nodes, or adding and removing taxonomies from the tree. Once a taxonomy has been added, any user who has access to the "Manage Categories" icon from his/her view can then browse the taxonomy tree and refer one of its nodes to the created documents.

```
<component>
   <key>org.exoplatform.services.cms.taxonomy.TaxonomyService</key>
   <type>org.exoplatform.services.cms.taxonomy.impl.TaxonomyServiceImpl</type>
   <init-params>
      <object-param>
         <name>defaultPermission.configuration</name>
                                                                  <object
 type="org.exoplatform.services.cms.taxonomy.impl.TaxonomyTreeDefaultUserPermission">
            <field name="permissions">
               <collection type="java.util.ArrayList">
                  <value>
```

```
                                                                                <object
type="org.exoplatform.services.cms.taxonomy.impl.TaxonomyTreeDefaultUserPermission$Permission">
                        <field name="identity">
                           <string>*:/platform/administrators</string>
                        </field>
                        <field name="read">
                           <string>true</string>
                        </field>
                        <field name="addNode">
                           <string>true</string>
                        </field>
                        <field name="setProperty">
                           <string>true</string>
                        </field>
                        <field name="remove">
                           <string>true</string>
                        </field>
                     </object>
                  </value>
                  <value>

                                                                                <object
type="org.exoplatform.services.cms.taxonomy.impl.TaxonomyTreeDefaultUserPermission$Permission">
                        <field name="identity">
                           <string>*:/platform/users</string>
                        </field>
                        <field name="read">
                           <string>true</string>
                        </field>
                        <field name="addNode">
                           <string>true</string>
                        </field>
                        <field name="setProperty">
                           <string>true</string>
                        </field>
                        <field name="remove">
                           <string>false</string>
                        </field>
                     </object>
                  </value>
               </collection>
            </field>
         </object>
```

```
      </object-param>
   </init-params>
</component>
```

Details:

- **Object** **type**: `org.exoplatform.services.cms.taxonomy.impl.TaxonomyTreeDefaultUserPermission`

Field	Type	Value	Description
permissions	`ArrayList`	*{java.util.ArrayList}*	The list of the default user permissions to access the taxonomy tree.

- **Object** **type**: `org.exoplatform.services.cms.taxonomy.impl.TaxonomyTreeDefaultUserPermission$Permission`

Field	Type	Description
identity	`String`	The name of user, group or membership.
read	`Boolean`	The permission to read the taxonomy tree.
addNode	`Boolean`	The permission to add a node to the taxonomy tree.
setProperty	`Boolean`	The permission to set properties for a node in the taxonomy tree.
remove	`Boolean`	The permission to remove a node from the taxonomy tree.

ThumbnailService

The *ThumbnailService* component is used to resize all the images into different sizes. Besides the default sizes, it also allows users to customize the images into the desired sizes. The configuration of this component is found in */core/core-configuration/src/main/webapp/WEB-INF/conf/wcm-core/core-services-configuration.xml*.

```
<component>
   <key>org.exoplatform.services.cms.thumbnail.ThumbnailService</key>
```

```
    <type>org.exoplatform.services.cms.thumbnail.impl.ThumbnailServiceImpl</type>
    <init-params>
      <value-param>
        <name>smallSize</name>
        <value>32x32</value>
      </value-param>
      <value-param>
        <name>mediumSize</name>
        <value>64x64</value>
      </value-param>
      <value-param>
        <name>largeSize</name>
        <value>300x300</value>
      </value-param>
      <value-param>
        <name>enable</name>
        <value>false</value>
      </value-param>
      <value-param>
        <name>mimetypes</name>
        <value>image/jpeg;image/png;image/gif;image/bmp</value>
      </value-param>
    </init-params>
</component>
```

Details:

- **Value-param**:

Name	Type	Value	Description
smallSize	`integer x integer`	`32x32`	The small thumbnail size.
mediumSize	`integer x integer`	`64x64`	The medium thumbnail size.
largeSize	`integer x integer`	`300x300`	The large thumbnail size.
enable	`Boolean`	`false`	Specify if the thumbnail is displayed or not.
mimetypes	`Images formats`	`image/jpeg;image/png;image/gif;image/bmp`	The image formats are supported.

TimelineService

The *TimelineService* component allows documents to be displayed by days, months and years. The configuration of this component is found in */core/core-configuration/src/main/webapp/WEB-INF/conf/wcm-core/core-services-configuration.xml*.

```
<component>
   <key>org.exoplatform.services.cms.timeline.TimelineService</key>
   <type>org.exoplatform.services.cms.timeline.impl.TimelineServiceImpl</type>
   <init-params>
      <value-param>
         <name>itemPerTimeline</name>
         <value>5</value>
      </value-param>
   </init-params>
</component>
```

Details:

- **Value-param**

Name	Type	Value	Description
itemPerTimeline	`integer`	`5`	The number of documents are displayed.

WatchDocumentService

The *WatchDocumentService* component allows users to watch/unwatch a document. If they are watching the document, they will receive a notification mail when there are any changes on the document. The configuration of this component is found in */core/core-configuration/src/main/webapp/WEB-INF/conf/wcm-core/core-services-configuration.xml*.

```
<component>
   <key>org.exoplatform.services.cms.watch.WatchDocumentService</key>
   <type>org.exoplatform.services.cms.watch.impl.WatchDocumentServiceImpl</type>
   <init-params>
      <object-param>
         <name>messageConfig</name>
         <description>description</description>
         <object type="org.exoplatform.services.cms.watch.impl.MessageConfig">
```

```
            <field name="sender">
               <string>support@exoplatform.com</string>
            </field>
            <field name="subject">
               <string>Your watching document is changed</string>
            </field>
            <field name="content">
               <string>The document that you are watching is changed.
                  Please go to ecm to see this change
               </string>
            </field>
         </object>
      </object-param>
   </init-params>
</component>
```

Details:

- **object-param**:
  - **Object type:** `org.exoplatform.services.cms.watch.impl.MessageConfig`

Field	Type	Value	Description
sender	`String`	`support@exoplatform.com`	The sender who sends the notification mail.
subject	`String`	`Your watching document is changed.`	The subject of the notification mail.
content	`String`	`The document that you are watching is changed. Please go to ecm to see this change.`	The content of the notification mail.

WCMService

The *WCMService* component allows setting expiration cache of portlets and checking given portals if they are shared portals or not. It also gets reference contents basing on item identifiers. The configuration of this component is found in */core/core-configuration/src/main/webapp/WEB-INF/conf/wcm-core/core-services-configuration.xml*.

```
<component>
```

```
  <key>org.exoplatform.services.wcm.core.WCMService</key>
  <type>org.exoplatform.services.wcm.core.impl.WCMServiceImpl</type>
  <init-params>
    <properties-param>
      <name>server.config</name>
      <description>server.config</description>
      <property name="expirationCache" value="30"/>
    </properties-param>
  </init-params>
</component>
```

Details:

Properties-param	Property name	Type	Value	Description
server.config	expirationCache	`integer`	`30`	The period in which the cache is clear in second. By default, the cache is cleared every 30 seconds.

External Component Plugins

This section describes about the main component plugins used in eXo Content. Each part supply an example configuration with the explanation about ini-params so you can know how to use these plugins.

AuthoringPublicationPlugin

This plugin is used to manage the publication lifecycle of web contents and DMS document on a portal page with more states and versions. The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/content-extended/authoring/configuration.xml*.

Sample configuration:

```
<component-plugin>
    <name>Authoring publication</name>
    <set-method>addPublicationPlugin</set-method>

 <type>org.exoplatform.services.wcm.extensions.publication.lifecycle.authoring.AuthoringPublicationPlugin
```

```
    </type>
     <description>This publication lifecycle publish a web content or DMS document to a portal page with more
       states and version.
    </description>
</component-plugin>
```

In which:

- **Name**: `Authoring publication`
- **Set-method**: `addPublicationPlugin`
- **Type**: `org.exoplatform.services.wcm.extensions.publication.lifecycle.authoring.AuthoringPublicationP`

BaseActionPlugin

This is an abstract class and the parent of ***ScriptActionPlugin*** and ***BPActionPlugin*** . You can create a link from this plugin to its children.

BPActionPlugin

This plugin is used to define the business process action in Workflow.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.cms.actions.ActionServiceContainer</target-component>
```

The configuration is applied mainly in *packaging/workflow/webapp/src/main/webapp/WEB-INF/workflow-extension/workflow/workflow-system-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
       <target-component>org.exoplatform.services.cms.actions.ActionServiceContainer</target-component>
   <component-plugin>
      <name>exo:businessProcessAction</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.plugin.actions.impl.BPActionPlugin</type>
      <priority>112</priority>
```

```
    <init-params>
      <object-param>
        <name>predefined.actions</name>
        <description>description</description>
        <object type="org.exoplatform.services.cms.actions.impl.ActionConfig">
          <field name="repository">
            <string>repository</string>
          </field>
          <field name="workspace">
            <string>collaboration</string>
          </field>
          <field name="actions">
            <collection type="java.util.ArrayList">
              <value>
              <object type="org.exoplatform.services.cms.actions.impl.ActionConfig$Action">
                <field name="type">
                  <string>exo:publishingProcess</string>
                </field>
                <field name="name">
                  <string>content publishing</string>
                </field>
                <field name="description">
                  <string>content publishing workflow</string>
                </field>
                <field name="srcWorkspace">
                  <string>collaboration</string>
                </field>
                <field name="srcPath">
                  <string>/Documents/Validation Requests</string>
                </field>
                <field name="isDeep">
                  <boolean>true</boolean>
                </field>
                <field name="lifecyclePhase">
                  <collection type="java.util.ArrayList">
                    <value>
                      <string>node_added</string>
                    </value>
                  </collection>
                </field>
                <field name="roles">
                  <string>*:/platform/users</string>
                </field>
                <field name="variables">
```

```
                        <string>
                                exo:supervisor=*:/organization/management/executive-
board;exo:validator=*:/platform/administrators
                        </string>
                    </field>
                    <field name="mixins">
                        <collection type="java.util.ArrayList">
                            <value>
                                                                                <object
 type="org.exoplatform.services.cms.actions.impl.ActionConfig$Mixin">
                                    <field name="name">
                                        <string>exo:publishLocation</string>
                                    </field>
                                    <field name="properties">
                                        <string>
                                            exo:publishWorkspace=collaboration;exo:publishPath=/
Documents/Live;exo:validator=*:/platform/administrators
                                        </string>
                                    </field>
                                </object>
                            </value>
                            <value>
                                                                                <object
 type="org.exoplatform.services.cms.actions.impl.ActionConfig$Mixin">
                                    <field name="name">
                                        <string>exo:pendingLocation</string>
                                    </field>
                                    <field name="properties">
                                        <string>
                                            exo:pendingWorkspace=collaboration;exo:pendingPath=/
Documents/Pending
                                        </string>
                                    </field>
                                </object>
                            </value>
                            <value>
                                                                                <object
 type="org.exoplatform.services.cms.actions.impl.ActionConfig$Mixin">
                                    <field name="name">
                                        <string>exo:backupLocation</string>
                                    </field>
                                    <field name="properties">
                                    <string>exo:backupWorkspace=backup;exo:backupPath=/Expired
                                        Documents
```

```
                                        </string>
                                    </field>
                                </object>
                            </value>
                            <value>
                                                                                <object
type="org.exoplatform.services.cms.actions.impl.ActionConfig$Mixin">
                                    <field name="name">
                                        <string>exo:trashLocation</string>
                                    </field>
                                    <field name="properties">
                                        <string>
                                exo:trashWorkspace=collaboration;exo:trashPath=/Documents/Trash
                                        </string>
                                    </field>
                                </object>
                            </value>
                            <value>
                                                                                <object
type="org.exoplatform.services.cms.actions.impl.ActionConfig$Mixin">
                                    <field name="name">
                                        <string>mix:affectedNodeTypes</string>
                                    </field>
                                    <field name="properties">
                                        <string>




exo:affectedNodeTypeNames=exo:article,exo:podcast,exo:sample,kfx:document,nt:file,rma:filePlan
                                        </string>
                                    </field>
                                </object>
                            </value>
                        </collection>
                    </field>
                </object>
            </value>
        </collection>
    </field>
</object>
</object-param>
</init-params>
</component-plugin>
```

```
</external-component-plugins>
```

In which:

- **Name**: `exo:businessProcessAction`
- **Set-method**: `addPlugin`
- **Type**: `org.exoplatform.services.plugin.actions.impl.BPActionPlugin`
- **Object type**: `org.exoplatform.services.cms.actions.impl.ActionConfig`

Field	Type	Value	Description
repository	`String`	`repository`	The repository name.
workspace	`String`	`collaboration`	The workspace name.
action	`ArrayList`	*{java.util.ArrayList}*	The action name.

ContentTypeFilterPlugin

This plugin is used to filter WCM node types.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.cms.templates.TemplateService</target-component>
```

The configuration is applied mainly in */packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/dms-extension/dms/dms-templates-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
        <target-component>org.exoplatform.services.cms.templates.TemplateService</target-component>
   <component-plugin>
      <name>FilterContentTypeForWCMSpecificFolder</name>
      <set-method>addContentTypeFilterPlugin</set-method>
      <type>org.exoplatform.services.cms.templates.ContentTypeFilterPlugin</type>
      <description>this plugin is used to filter wcm nodetype</description>
      <init-params>
```

```
        <object-param>
          <name>cssFolderFilter</name>
          <description>only exo:cssFile can be created in exo:cssFolder</description>
                                                                                <object
type="org.exoplatform.services.cms.templates.ContentTypeFilterPlugin$FolderFilterConfig">
            <field name="folderType">
              <string>exo:cssFolder</string>
            </field>
            <field name="contentTypes">
              <collection type="java.util.ArrayList">
                <value>
                  <string>exo:cssFile</string>
                </value>
              </collection>
            </field>
          </object>
        </object-param>
        <object-param>
          ...
        </object-param>
      </init-params>
    </component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `FilterContentTypeForWCMSpecificFolder`
- **Set-method**: `addContentTypeFilterPlugin`
- **Type**: `org.exoplatform.services.cms.templates.ContentTypeFilterPlugin`
- **Object** **type**: `org.exoplatform.services.cms.templates.ContentTypeFilterPlugin$FolderFilterConfig`

Field	Type	Value	Description
folderType	`String`	`exo:cssFolder`	The folder type.
contentTypes	`ArrayList`	*{java.util.ArrayList}*	The content type.

ContextPlugin

This plugin is used to store the context configuration of a publication lifecycle. To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.wcm.extensions.publication.PublicationManager</
target-component>
```

The configuration is applied mainly in *packaging/ecmdemo/webapp/src/main/webapp/WEB-INF/conf/sample-portal/wcm/content-configuration.xml* or *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/content-extended/authoring/configuration.xml*.

Sample configuration:

```
<external-component-plugins>
                                                                                <target-
component>org.exoplatform.services.wcm.extensions.publication.PublicationManager</target-
component>
   <component-plugin>
      <name>AddContext</name>
      <set-method>addContext</set-method>
      <type>org.exoplatform.services.wcm.extensions.publication.context.ContextPlugin</type>
      <init-params>
         <object-param>
            <name>contexts</name>
                                                                                    <object
 type="org.exoplatform.services.wcm.extensions.publication.context.impl.ContextConfig">
               <field name="contexts">
                  <collection type="java.util.ArrayList">
                     <value>

                                                                                <object
 type="org.exoplatform.services.wcm.extensions.publication.context.impl.ContextConfig$Context">
                           <field name="name">
                              <string>context2</string>
                           </field>
                           <field name="priority">
                              <string>100</string>
                           </field>
                           <field name="lifecycle">
                              <string>lifecycle2</string>
                           </field>
                           <field name="site">
                              <string>acme</string>
                           </field>
                        </object>
```

```
                    </value>
                </collection>
            </field>
        </object>
    </object-param>
</init-params>
</component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `AddContext`
- **Set-method**: `addContext`
- **Type**: `org.exoplatform.services.wcm.extensions.publication.context.ContextPlugin`
- **Object type**: `org.exoplatform.services.wcm.extensions.publication.context.impl.ContextConfig`

Field	Type	Value	Description
name	`string`	`context2`	The name of the context.
priority	`string`	`100`	The context priority, the higher number indicates higher priority. Because a site may have several lifecycles, the lifecycle with higher priority will be executed sooner.
lifecycle	`string`	`lifecycle2`	The name of the lifecycle.
site	`string`	`acme`	The site that will apply the context configuration.

CreatePortalPlugin

This is an abstract class and the parent of ***axonomyPlugin*** . You can create a link from this plugin to its children.

ExcludeIncludeDataTypePlugin

This plugin is used in the *SiteSearchService* component to filter the search results before these results are presented on the search page.

The configuration is applied mainly in *core/core-configuration/src/main/webapp/WEB-INF/conf/wcm-core/core-search-configuration.xml*.

Sample configuration:

```
<component-plugins>
  <component-plugin>
    <name>ExcludeMimeTypes</name>
    <set-method>addExcludeIncludeDataTypePlugin</set-method>
    <type>org.exoplatform.services.wcm.search.ExcludeIncludeDataTypePlugin</type>
    <init-params>
      <properties-param>
        <name>search.exclude.datatypes</name>
        <description>exclude some data type when search</description>
          <property name="mimetypes" value="text/css,text/javascript,application/x-javascript,text/ecmascript"/>
      </properties-param>
    </init-params>
  </component-plugin>
</component-plugins>
```

In which:

- **Name**: `ExcludeMimeTypes`
- **Set-method**: `addExcludeIncludeDataTypePlugin`
- **Type**: `org.exoplatform.services.wcm.search.ExcludeIncludeDataTypePlugin`
- The plugin has the following parameter:

Properties-param	Description
search.exclude.datatype	Exclude some data types when doing search.

- The `search.exclude.datatype` property includes two attributes:

Attribute	Value	Description
name	`mimetypes`	The name of the property param.
value	`text/css,text/javascript,application/x-javascript,text/ecmascript`	The list of mimetypes which will be excluded from the search results.

FriendlyPlugin

This plugin is used to refine URLs in WCM.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.wcm.friendly.FriendlyService</target-component>
```

The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/content-extended/friendly/configuration.xml*.

Sample configuration:

```
<external-component-plugins>
                <target-component>org.exoplatform.services.wcm.friendly.FriendlyService</target-component>
   <component-plugin>
      <name>FriendlyService.addConfiguration</name>
      <set-method>addConfiguration</set-method>
      <type>org.exoplatform.services.wcm.friendly.impl.FriendlyPlugin</type>
      <description>Configures</description>
      <priority>100</priority>
      <init-params>
         <value-param>
            <name>enabled</name>
            <value>true</value>
         </value-param>
         <object-param>
            <name>friendlies.configuration</name>
            <object type="org.exoplatform.services.wcm.friendly.impl.FriendlyConfig">
               <field name="friendlies">
                  <collection type="java.util.ArrayList">
                     <value>
```

```
                                                                    <object
type="org.exoplatform.services.wcm.friendly.impl.FriendlyConfig$Friendly">
                            <field name="friendlyUri">
                               <string>documents</string>
                            </field>
                            <field name="unfriendlyUri">
                              <string>/public/acme/detail?content-id=/repository/collaboration</string>
                            </field>
                          </object>
                        </value>
                        <value>
                                                                    <object
type="org.exoplatform.services.wcm.friendly.impl.FriendlyConfig$Friendly">
                            <field name="friendlyUri">
                               <string>files</string>
                            </field>
                            <field name="unfriendlyUri">
                               <string>/rest-ecmdemo/jcr/repository/collaboration</string>
                            </field>
                          </object>
                        </value>
                      </collection>
                    </field>
                  </object>
                </object-param>
              </init-params>
            </component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `FriendlyService.addConfiguration`
- **Set-method**: `addConfiguration`
- **Type**: `org.exoplatform.services.wcm.friendly.impl.FriendlyPlugin`
- **Object type**: `org.exoplatform.services.wcm.friendly.impl.FriendlyConfig`

Field	Type	Value	Description
friendlyUrl	`string`	`documents`	The object that will be applied the friendly URL.

Field	Type	Value	Description
unfriendlyUrl	`string`	`/public/acme/detail?content-id=/repository/collaboration`	The path to the location that will be applied the friendly URL.

ImageThumbnailPlugin

This plugin is used to configure the file types and get thumbnail for images.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.cms.thumbnail.ThumbnailService</target-component>
```

The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/dms-extension/dms/dms-thumbnail-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
          <target-component>org.exoplatform.services.cms.thumbnail.ThumbnailService</target-component>
   <component-plugin>
      <name>ImageThumbnailPlugin</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.cms.thumbnail.impl.ImageThumbnailPlugin</type>
      <init-params>
         <object-param>
            <name>thumbnailType</name>
            <description>Thumbnail types</description>
            <object type="org.exoplatform.services.cms.thumbnail.impl.ThumbnailType">
               <field name="mimeTypes">
                  <collection type="java.util.ArrayList">
                     <value>
                        <string>image/jpeg</string>
                     </value>
                     <value>
                        <string>image/png</string>
                     </value>
                     <value>
```

```
                        <string>image/gif</string>
                    </value>
                    <value>
                        <string>image/bmp</string>
                    </value>
                    <value>
                        <string>image/tiff</string>
                    </value>
                </collection>
            </field>
        </object>
    </object-param>
</init-params>
</component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `ImageThumbnailPlugin`
- **Set-method**: `addPlugin`
- **Type**: `org.exoplatform.services.cms.thumbnail.impl.ImageThumbnailPlugin`
- **Object type**: `org.exoplatform.services.cms.thumbnail.impl.ThumbnailType`

Field	Type	Value	Description

mimeTypes	`String`	`image/jpeg`
		`image/png`
		`image/gif`
		`image/bmp`
		`image/tiff`
			The list of thumbnail image types.

InitialWebcontentPlugin

When a portal is created, this plugin will deploy initial web-contents as the site artifact into the **Site Artifact** folder of that portal.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.wcm.portal.artifacts.CreatePortalArtifactsService</target-component>
```

The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/wcm-extension/wcm/newsletter-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
                                                                                <target-component>org.exoplatform.services.wcm.portal.artifacts.CreatePortalArtifactsService</target-component>
   <component-plugin>
      <name>Initial webcontent artifact for each site</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.wcm.webcontent.InitialWebContentPlugin</type>
      <description>This plugin deploy some initial webcontent as site artifact to site artifact folder of portal when
         a portal is
         created
      </description>
      <init-params>
         <object-param>
            <name>Portal logo data</name>
            <description>Deployment Descriptor</description>
            <object type="org.exoplatform.services.deployment.DeploymentDescriptor">
               <field name="target">
               <object type="org.exoplatform.services.deployment.DeploymentDescriptor$Target">
                     <field name="repository">
                        <string>repository</string>
                     </field>
                     <field name="workspace">
                        <string>collaboration</string>
                     </field>
                     <field name="nodePath">
                        <string>/sites content/live/{portalName}/web contents/site artifacts</string>
                     </field>
                  </object>
               </field>
```

```
                <field name="sourcePath">
                        <string>war:/conf/sample-portal/wcm/artifacts/site-resources/acme-templates/
Logo.xml</string>
                </field>
              </object>
          </object-param>
          <object-param>
              <name>Portal signin data</name>
              <description>Deployment Descriptor</description>
              <object type="org.exoplatform.services.deployment.DeploymentDescriptor">
                 <field name="target">
                 <object type="org.exoplatform.services.deployment.DeploymentDescriptor$Target">
                        <field name="repository">
                           <string>repository</string>
                        </field>
                        <field name="workspace">
                           <string>collaboration</string>
                        </field>
                        <field name="nodePath">
                           <string>/sites content/live/{portalName}/web contents/site artifacts</string>
                        </field>
                     </object>
                </field>
                <field name="sourcePath">
                        <string>war:/conf/sample-portal/wcm/artifacts/site-resources/acme-templates/
Signin.xml</string>
                </field>
              </object>
          </object-param>
          <object-param>
              ...
          </object-param>
       </init-params>
    </component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `AddLifecycle`
- **Set-method**: `addPlugin`
- **Type**: `org.exoplatform.services.wcm.webcontent.InitialWebContentPlugin`
- **Object type**: `org.exoplatform.services.deployment.DeploymentDescriptor$Target`

Name	Type	Value	Description
repository	`string`	`repository`	The repository into which the initial web contents will be deployed.
workspace	`string`	`collaboration`	The workspace into which the initial web contents will be deployed.
nodePath	`string`	`/sites content/live//web contents/site artifacts`	The target node where the initial web-contents will be deployed into.
sourcePath	`string`	`war:/conf/sample-portal/wcm/artifacts/site-resources/acme-templates/Logo.xml`	The path to the source that this plugin will get data.

LinkDeploymentPlugin

This plugin is used to create predefined Symlinks into the system.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.deployment.WCMContentInitializerService</
target-component>
```

The configuration is applied mainly in *packaging/ecmdemo/webapp/src/main/webapp/WEB-INF/conf/sample-portal/wcm/deployment/acme-deployment-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
        <target-component>org.exoplatform.services.deployment.WCMContentInitializerService</
target-component>
    <component-plugin>
        <name>Content Initializer Service</name>
        <set-method>addPlugin</set-method>
        <type>org.exoplatform.services.deployment.plugins.LinkDeploymentPlugin</type>
```

```
    <description>Link Deployment Plugin</description>
    <init-params>
      <object-param>
        <name>link01</name>
        <description>Deployment Descriptor</description>
       <object type="org.exoplatform.services.deployment.plugins.LinkDeploymentDescriptor">
          <field name="sourcePath">
              <string>repository:collaboration:/sites content/live/acme/web contents/News/
News1</string>
          </field>
          <field name="targetPath">
            <string>repository:collaboration:/sites content/live/acme/categories/acme</string>
          </field>
        </object>
      </object-param>
      <object-param>
        <name>link02</name>
        <description>Deployment Descriptor</description>
       <object type="org.exoplatform.services.deployment.plugins.LinkDeploymentDescriptor">
          <field name="sourcePath">
              <string>repository:collaboration:/sites content/live/acme/web contents/News/
News2</string>
          </field>
          <field name="targetPath">
            <string>repository:collaboration:/sites content/live/acme/categories/acme</string>
          </field>
        </object>
      </object-param>
      <object-param>
        <name>link03</name>
        <description>Deployment Descriptor</description>
       <object type="org.exoplatform.services.deployment.plugins.LinkDeploymentDescriptor">
          <field name="sourcePath">
              <string>repository:collaboration:/sites content/live/acme/web contents/News/
News3</string>
          </field>
          <field name="targetPath">
            <string>repository:collaboration:/sites content/live/acme/categories/acme</string>
          </field>
        </object>
      </object-param>
    </init-params>
  </component-plugin>
```

```
</external-component-plugins>
```

In which:

- **Name**: `Content Initializer Service`
- **Set-method**: `addPlugin`
- **Type**: `org.exoplatform.services.wcm.webcontent.InitialWebContentPlugin`
- **Object type**: `org.exoplatform.services.deployment.plugins.LinkDeploymentDescriptor`

Field	Type	Value	Description
sourcePath	`string`	`repository:collaboration:/sites content/live/acme/web contents/News/News1`	The path to the source where this plugin will get data.
targetPath	`string`	`repository:collaboration:/sites content/live/acme/categories/acme`	The path to the target where this plugin will deploy.

LockGroupsOrUsersPlugin

This plugin is used to configure predefined groups or users for lock administration. To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.cms.lock.LockService</target-component>
```

The configuration is applied mainly in *core/core-configuration/src/main/webapp/WEB-INF/conf/wcm-core/core-services-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
  <target-component>org.exoplatform.services.cms.lock.LockService</target-component>
  <component-plugin>
    <name>predefinedLockGroupsOrUsersPlugin</name>
    <set-method>addLockGroupsOrUsersPlugin</set-method>
```

```
    <type>org.exoplatform.services.cms.lock.impl.LockGroupsOrUsersPlugin</type>
    <init-params>
      <object-param>
        <name>LockGroupsOrUsers.configuration</name>
      <description>configuration predefined groups or users for lock administrator</description>
        <object type="org.exoplatform.services.cms.lock.impl.LockGroupsOrUsersConfig">
          <field name="settingLockList">
            <collection type="java.util.ArrayList">
              <value>
                <string>*:/platform/administrators</string>
              </value>
            </collection>
          </field>
        </object>
      </object-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `predefinedLockGroupsOrUsersPlugin`
- **Set-method**: `addLockGroupsOrUsersPlugin`
- **Type**: `org.exoplatform.services.cms.lock.impl.LockGroupsOrUsersPlugin`
- **Object type**: `org.exoplatform.services.cms.lock.impl.LockGroupsOrUsersConfig`

Field	Type	Value	Description
settingLockList	`ArrayList`	*{java.util.ArrayList}*	The list of the groups or user to be locked.

ManageDrivePlugin

This plugin is used to create a predefined drive into a repository. A drive can be considered as a shortcut in the content repository, a quick access to some places for users. You can restrict the visibility of this drive to a group/user and apply a specific view depending on the content you have in this area.

A drive is the combination of:

- Path: the root folder of the drive.
- View: how we can see contents, such as by list, thumbnails, coverflow.
- Role: the visibility to every users, a group or a single user.

- Options: allow you to specify whether to see hidden nodes or not and to create folders in this drive or not.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.cms.drives.ManageDriveService</target-component>
```

The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/dms-extension/dms/dms-drives-configuration.xml*.

The following structure is used for drives configuration.

```
<external-component-plugins>
            <target-component>org.exoplatform.services.cms.drives.ManageDriveService</target-component>
   <component-plugin>
      <name>manage.drive.plugin</name>
      <set-method>setManageDrivePlugin</set-method>
      <type>org.exoplatform.services.cms.drives.impl.ManageDrivePlugin</type>
      <description>Nothing</description>
      <init-params>
         <object-param>
            There are initializing attributes of org.exoplatform.services.cms.drives.DriveData object
         </object-param>
      </init-params>
   </component-plugin>
</external-component-plugins>
```

The file that contains the structure above will be configured in the *configuration.xml* file as the following:

```
<import>war:/conf/wcm-extension/dms/drives-configuration.xml</import>
```

Sample configuration:

```
<external-component-plugins>
```

```
            <target-component>org.exoplatform.services.cms.drives.ManageDriveService</target-
component>
   <component-plugin>
      <name>manage.drive.plugin</name>
      <set-method>setManageDrivePlugin</set-method>
      <type>org.exoplatform.services.cms.drives.impl.ManageDrivePlugin</type>
      <description>Nothing</description>
      <init-params>
         <object-param>
            <name>Managed Sites</name>
            <description>Managed Sites</description>
            <object type="org.exoplatform.services.cms.drives.DriveData">
               <field name="name">
                  <string>Managed Sites</string>
               </field>
               <field name="repository">
                  <string>repository</string>
               </field>
               <field name="workspace">
                  <string>collaboration</string>
               </field>
               <field name="permissions">
                  <string>*:/platform/administrators</string>
               </field>
               <field name="homePath">
                  <string>/sites content/live</string>
               </field>
               <field name="icon">
                  <string/>
               </field>
               <field name="views">
                  <string>wcm-view</string>
               </field>
               <field name="viewPreferences">
                  <boolean>false</boolean>
               </field>
               <field name="viewNonDocument">
                  <boolean>true</boolean>
               </field>
               <field name="viewSideBar">
                  <boolean>true</boolean>
               </field>
               <field name="showHiddenNode">
                  <boolean>false</boolean>
```

```
            </field>
            <field name="allowCreateFolders">
               <string>nt:folder,nt:unstructured</string>
            </field>
            <field name="allowNodeTypesOnTree">
               <string>*</string>
            </field>
         </object>
      </object-param>
      <object-param>
         <name>Public</name>
         <description>Public drive</description>
         <object type="org.exoplatform.services.cms.drives.DriveData">
            <field name="name">
               <string>Public</string>
            </field>
            <field name="repository">
               <string>repository</string>
            </field>
            <field name="workspace">
               <string>collaboration</string>
            </field>
            <field name="permissions">
               <string>*:/platform/users</string>
            </field>
            <field name="homePath">
               <string>/Users/${userId}/Public</string>
            </field>
            <field name="icon">
               <string/>
            </field>
            <field name="views">
               <string>simple-view, admin-view</string>
            </field>
            <field name="viewPreferences">
               <boolean>false</boolean>
            </field>
            <field name="viewNonDocument">
               <boolean>false</boolean>
            </field>
            <field name="viewSideBar">
               <boolean>true</boolean>
            </field>
            <field name="showHiddenNode">
```

```
                <boolean>false</boolean>
              </field>
              <field name="allowCreateFolders">
                <string>nt:folder,nt:unstructured</string>
              </field>
              <field name="allowNodeTypesOnTree">
                <string>*</string>
              </field>
            </object>
          </object-param>
        </init-params>
    </component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `manage.drive.plugin`
- **Set-method**: `setManageDrivePlugin`
- **Type**: `org.exoplatform.services.cms.drives.impl.ManageDrivePlugin`
- **Object type**: `org.exoplatform.services.cms.drives.DriveData`

Field	Type	Value	Description
name	`String`	`Public`	The name of drive which must be unique.
repository	`String`	`repository`	Content Repository where to find the root path.
workspace	`String`	`collaboration`	Workspace in the Content Repository.
homePath	`String`	`/sites content/live`	Root path in the Content Repository. **`userId`** can be used to use the userId at runtime in the path.
permissions	`String`	*`*:/platform/administrators`*	Visibility of the drive based on eXo rights. For example: `*:/platform/users`
icon	`String`	N/A	The Url to the icon.
views	`String`	`wcm-view`	The list of views you want

Field	Type	Value	Description
			to use, separated by commas. For example: `simple-view,admin-view`
viewPreferences	`Boolean`	`false`	The **User Preference** icon will be visible if true.
viewNonDocument	`Boolean`	`true`	Non-document types will be visible in the user view if true.
viewSideBar	`Boolean`	`true`	Show/Hide the left bar (with navigation and filters).
showHiddenNode	`Boolean`	`false`	Hidden nodes will be visible if true.
allowCreateFolders	`String`	`nt:folder,nt:unstructured`	List of node types that you can create as folders. For example: `nt:folder,nt:unstructured`.
allowNodeTypesOnTree	`String`	*	Allow you to filter node types in the navigation tree. For example, the default value is "*" to show all content types.

ManageViewPlugin

This plugin is used to create a predefined View into a repository. A View can include many object parameters. Parameters are used to create default Views, Templates and Actions of **Manage View** service. View enables administrators to customize View classification that can impact on users in exploring workspace. Each object-param has a type that is a class representing all properties of a View.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.cms.views.ManageViewService</target-component>
```

The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/dms-extension/dms/dms-views-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
            <target-component>org.exoplatform.services.cms.views.ManageViewService</target-component>
   <component-plugin>
      <name>manage.view.plugin</name>
      <set-method>setManageViewPlugin</set-method>
      <type>org.exoplatform.services.cms.views.impl.ManageViewPlugin</type>
      <description>this plugin manage user view</description>
      <init-params>
         <value-param>
            <name>autoCreateInNewRepository</name>
            <value>true</value>
         </value-param>
         <value-param>
            <name>predefinedViewsLocation</name>
            <value>war:/conf/dms-extension/dms/artifacts</value>
         </value-param>
         <value-param>
            <name>repository</name>
            <value>repository</value>
         </value-param>
         <object-param>
            <name>System-View</name>
            <description>View configuration for System workspace</description>
            <object type="org.exoplatform.services.cms.views.ViewConfig">
               <field name="name">
                  <string>system-view</string>
               </field>
               <field name="permissions">
                  <string>*:/platform/administrators</string>
               </field>
               <field name="template">
                  <string>/exo:ecm/views/templates/ecm-explorer/SystemView</string>
               </field>
               <field name="tabList">
                  <collection type="java.util.ArrayList">
                     <value>
                        <object type="org.exoplatform.services.cms.views.ViewConfig$Tab">
```

```
                        <field name="tabName">
                            <string>Info</string>
                        </field>
                        <field name="buttons">
                                    <string>viewNodeType; viewPermissions; viewProperties;
showJCRStructure</string>
                        </field>
                    </object>
                </value>
            </collection>
        </field>
    </object>
</object-param>
<object-param>
    <name>System Template</name>
    <description>Template for display documents in list style</description>
    <object type="org.exoplatform.services.cms.views.TemplateConfig">
        <field name="type">
            <string>ecmExplorerTemplate</string>
        </field>
        <field name="name">
            <string>SystemView</string>
        </field>
        <field name="warPath">
            <string>/ecm-explorer/SystemView.gtmpl</string>
        </field>
    </object>
</object-param>
</init-params>
</component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `manage.view.plugin`
- **Set-method**: `setManageViewPlugin`
- **Type**: `org.exoplatform.services.cms.views.impl.ManageViewPlugin`
- **Init-param**:

value-param	Type	Value	Description
autoCreateInNewRepository	`boolean`	`true`	

value-param	Type	Value	Description
			Allow creating a predefined View in this repository if the value is "true".
predefinedViewsLocation	`String`	war:/conf/dms-extension/dms/artifacts	The location of the View node in the repository.
repository	`String`	`repository`	The repository name.

- **Object type**: `org.exoplatform.services.cms.views.ViewConfig`

Field	Type	Value	Description
name	`String`	`system-view`	The name of view which must be unique inside WCM.
permissions	`String`	**:/platform/administrators*	Visibility of the view based on eXo rights.
template	`String`	`/exo:ecm/views/templates/ecm-explorer/SystemView`	Specify path to the template location.
tabList	`ArrayList`	*{java.util.ArrayList}*	Include a set of view names.

- **Object type**: `org.exoplatform.services.cms.views.ViewConfig$Tab`

Field	Type	Value	Description
tabName	`String`	`Info`	The name of tab which must be unique.
button	`String`	`viewNodeType; viewPermissions; viewProperties; showJCRStructure`	Specify a set of view component names.

- **Object type**: `org.exoplatform.services.cms.views.TemplateConfig`

Field	Type	Vsalue	Description
type	`String`	`ecmExplorerTemplate`	Specify if a name is truly a class representing all properties of a view.

Field	Type	Vsalue	Description
name	`String`	`system-view`	Specify a set of view component names.
warPath	`String`	`/ecm-explorer/SystemView.gtmpl`	Specify a template location to view.

PDFThumbnailPlugin

This plugin is to set the supported file types of PDF thumbnail. See also ***ImageThumbnailPlugin*** .

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.cms.thumbnail.ThumbnailService</target-component>
```

The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/dms-extension/dms/dms-thumbnail-configuration.xml*.

Sample configuration:

```
<component-plugin>
   <name>PDFThumbnailPlugin</name>
   <set-method>addPlugin</set-method>
   <type>org.exoplatform.services.cms.thumbnail.impl.PDFThumbnailPlugin</type>
   <init-params>
      <object-param>
         <name>thumbnailType</name>
         <description>Thumbnail types</description>
         <object type="org.exoplatform.services.cms.thumbnail.impl.ThumbnailType">
            <field name="mimeTypes">
               <collection type="java.util.ArrayList">
                  <value>
                     <string>application/pdf</string>
                  </value>
               </collection>
            </field>
         </object>
      </object-param>
   </init-params>
</component-plugin>
```

In which:

- **Name**: `PDFThumbnailPlugin`
- **Set-method**: `addPlugin`
- **Type**: `org.exoplatform.services.cms.thumbnail.impl.PDFThumbnailPlugin`
- **Object type**: `org.exoplatform.services.cms.thumbnail.impl.ThumbnailType`

Field	Type	Value	Description
mimeTypes	`String`	`application/pdf`	The MIME type of the pdf thumbnail.

PorletTemplatePlugin

This plugin is used to import the view templates into Content List Viewer.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.cms.views.ApplicationTemplateManagerService</target-component>
```

The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/wcm-extension/dms/application-templates-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
                                                                    <target-component>org.exoplatform.services.cms.views.ApplicationTemplateManagerService</target-component>
   <component-plugin>
      <name>clv.templates.plugin</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.cms.views.PortletTemplatePlugin</type>
         <description>This plugin is used to import views templates for Content List Viewer</description>
      <init-params>
         <value-param>
            <name>portletName</name>
```

```
            <value>content-list-viewer</value>
          </value-param>
          <value-param>
            <name>portlet.template.path</name>
            <value>war:/conf/wcm-artifacts/application-templates/content-list-viewer</value>
          </value-param>
          <object-param>
            <name>default.folder.list.viewer</name>
            <description>Default folder list viewer groovy template</description>
                                                                                <object
 type="org.exoplatform.services.cms.views.PortletTemplatePlugin$PortletTemplateConfig">
                <field name="templateName">
                  <string>UIContentListPresentationDefault.gtmpl</string>
                </field>
                <field name="category">
                  <string>list</string>
                </field>
              </object>
          </object-param>
          <object-param>
            ....
          </object-param>
        </init-params>
    </component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `clv.templates.plugin`
- **Set-method**: `addPlugin`
- **Type**: `org.exoplatform.services.cms.views.PortletTemplatePlugin`
- **Init-param**:

Value-param	Type	Value	Description
portletName	`String`	`content-list-viewer`	The name of the portlet.
portlet.template.path	`String`	`war:/conf/wcm-artifacts/application-templates/content-list-viewer`	The path to the configuration of the portlet.

- **Object** **type**: *org.exoplatform.services.cms.views.PortletTemplatePlugin$PortletTemplateConfig*

Field	Type	Description
templateName	`string`	The name of the GROOVY template.
category	`string`	The category name.

PredefinedProcessesPlugin

This plugin is used to import the predefined processes into the system.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.workflow.WorkflowServiceContainer</target-component>
```

The configuration is applied mainly in *packaging/workflow/webapp/src/main/webapp/WEB-INF/workflow-extension/workflow/bonita-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
        <target-component>org.exoplatform.services.workflow.WorkflowServiceContainer</target-component>
    <component-plugin>
        <name>deploy.predefined.processes</name>
        <set-method>addPlugin</set-method>
        <type>org.exoplatform.services.workflow.PredefinedProcessesPlugin</type>
        <init-params>
            <object-param>
                <name>predefined.processes</name>
                <description>load of default business processes</description>
                <object type="org.exoplatform.services.workflow.ProcessesConfig">
                    <field name="processLocation">
                        <string>war:/conf/bp</string>
                    </field>
                    <field name="predefinedProcess">
                        <collection type="java.util.HashSet">
                            <value>
                                <string>/exo-ecms-ext-workflow-bp-bonita-content-2.3.2.jar</string>
```

```
                    </value>
                    <value>
                      <string>/exo-ecms-ext-workflow-bp-bonita-payraise-2.3.2.jar</string>
                    </value>
                    <value>
                      <string>/exo-ecms-ext-workflow-bp-bonita-holiday-2.3.2.jar</string>
                    </value>
                  </collection>
                </field>
              </object>
            </object-param>
          </init-params>
        </component-plugin>
</external-component-plugins>
```

In which:

- **name**: `deploy.predefined.processes`
- **set-method**: `addPlugin`
- **type**: `org.exoplatform.services.workflow.PredefinedProcessesPlugin`
- **Object type**: `org.exoplatform.services.workflow.ProcessesConfig`

Field	Type	Value	Description
processLocation	`string`	`war:/conf/bp`	The path to the process.
predefinedProcess	`HashSet`	*{java.util.HashSet}*	The list of the processes.

PublicationPlugin

This is an abstract class and the parent of ***StageAndVersionPublicationPlugin*** and ***AuthoringPublicationPlugin*** . You can create a link from this plugin to its children.

QueryPlugin

This plugin is used to store predefined queries into the repositories of the system.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.cms.queries.QueryService</target-component>
```

The configuration is applied mainly in */packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/dms-extension/dms/dms-queries-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
   <target-component>org.exoplatform.services.cms.queries.QueryService</target-component>
   <component-plugin>
      <name>query.plugin</name>
      <set-method>setQueryPlugin</set-method>
      <type>org.exoplatform.services.cms.queries.impl.QueryPlugin</type>
      <description>Nothing</description>
      <init-params>
         <value-param>
            <name>autoCreateInNewRepository</name>
            <value>true</value>
         </value-param>
         <value-param>
            <name>repository</name>
            <value>repository</value>
         </value-param>
         <object-param>
            <name>CreatedDocuments</name>
            <description>documents created by the current user</description>
            <object type="org.exoplatform.services.cms.queries.impl.QueryData">
               <field name="name">
                  <string>Created Documents</string>
               </field>
               <field name="language">
                  <string>xpath</string>
               </field>
               <field name="statement">
                  <string>//*[(@jcr:primaryType = 'exo:article' or @jcr:primaryType = 'nt:file') and
                     @exo:owner='${UserId}$'] order by @exo:dateCreated descending
                  </string>
               </field>
               <field name="permissions">
                  <collection type="java.util.ArrayList">
                     <value>
                        <string>*:/platform/users</string>
                     </value>
                  </collection>
               </field>
```

```
                <field name="cachedResult">
                  <boolean>false</boolean>
                </field>
              </object>
            </object-param>
            <object-param>
              <name>CreatedDocumentsDayBefore</name>
              <description>documents created the day before</description>
              <object type="org.exoplatform.services.cms.queries.impl.QueryData">
                <field name="name">
                  <string>CreatedDocumentDayBefore</string>
                </field>
                <field name="language">
                  <string>xpath</string>
                </field>
                <field name="statement">
                    <string>//element(*,exo:article)[@exo:dateCreated &lt; xs:dateTime('${Date}$')]
order by
                      @exo:dateCreated descending
                  </string>
                </field>
                <field name="permissions">
                  <collection type="java.util.ArrayList">
                    <value>
                      <string>*:/platform/users</string>
                    </value>
                  </collection>
                </field>
                <field name="cachedResult">
                  <boolean>true</boolean>
                </field>
              </object>
            </object-param>
            <object-param>
              <name>AllArticles</name>
              <description>All articles</description>
              <object type="org.exoplatform.services.cms.queries.impl.QueryData">
                <field name="name">
                  <string>All Articles</string>
                </field>
                <field name="language">
                  <string>xpath</string>
                </field>
                <field name="statement">
```

```
                <string>//element(*,exo:article) order by @exo:dateCreated descending</string>
              </field>
              <field name="permissions">
                <collection type="java.util.ArrayList">
                  <value>
                    <string>*:/platform/users</string>
                  </value>
                </collection>
              </field>
              <field name="cachedResult">
                <boolean>true</boolean>
              </field>
            </object>
          </object-param>
        </init-params>
    </component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `predefinedTaxonomyPlugin`
- **Set-method**: `setQueryPlugin`
- **Type**: `org.exoplatform.services.cms.queries.impl.QueryPlugin`
- **Init-param**:

Value-param	Type	Value	Description
`autoCreateInNewRepository`	`boolean`	`true`	Store queries in a new repository if the value is "true".
`repository`	`string`	`repository`	The repository to the target node.

- **Object type**: `org.exoplatform.services.cms.queries.impl.QueryData`

Field	Type	Description
name	`String`	The name of the query.
language	`String`	The language of the query (Xpath, SQL).
statement	`String`	The query statement.
permissions	`ArrayList`	The permission which users must have to use this query.

Field	Type	Description
cachedResult	`Boolean`	Specify if the query is cached or not.

RemovePortalPlugin

This is an abstract class and the parent of ***RemoveTaxonomyPlugin*** . You can create a link from this plugin to its children.

RemoveTaxonomyPlugin

This plugin is used to invalidate taxonomy trees in **categories** folder of a portal when the portal is removed.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-
component>org.exoplatform.services.wcm.portal.artifacts.RemovePortalArtifactsService</
target-component>
```

Sample configuration:

```
<external-component-plugins>
                                                                    <target-
component>org.exoplatform.services.wcm.portal.artifacts.RemovePortalArtifactsService</
target-component>
   <component-plugin>
      <name>Remove taxonomy tree</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.wcm.category.RemoveTaxonomyPlugin</type>
     <description>This plugin invalidate taxonomy tree to categories folder of portal when a portal
 is removed
      </description>
   </component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `Remove taxonomy tree`
- **Set-method**: `addPlugin`

- **Type**: `org.exoplatform.services.wcm.category.RemoveTaxonomyPlugin`

ScriptActionPlugin

This plugin is used to import the predefined script actions into the system.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.cms.actions.ActionServiceContainer</target-component>
```

The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/dms-extension/dms/dms-actions-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
        <target-component>org.exoplatform.services.cms.actions.ActionServiceContainer</target-component>
    <component-plugin>
        <name>exo:scriptAction</name>
        <set-method>addPlugin</set-method>
        <type>org.exoplatform.services.cms.actions.impl.ScriptActionPlugin</type>
        <init-params>
            <object-param>
                <name>predefined.actions</name>
                <description>description</description>
                <object type="org.exoplatform.services.cms.actions.impl.ActionConfig">
                    <field name="repository">
                        <string>repository</string>
                    </field>
                    <field name="workspace">
                        <string>collaboration</string>
                    </field>
                    <field name="actions">
                        <collection type="java.util.ArrayList">
                            <value>
                            <object type="org.exoplatform.services.cms.actions.impl.ActionConfig$Action">
                                <field name="type">
                                    <string>exo:sendMailAction</string>
                                </field>
                                <field name="name">
```

```
                        <string>sendMail</string>
                    </field>
                    <field name="description">
                        <string>send a notification mail</string>
                    </field>
                    <field name="srcWorkspace">
                        <string>collaboration</string>
                    </field>
                    <field name="srcPath">
                        <string>/Documents/Validation Requests</string>
                    </field>
                    <field name="isDeep">
                        <boolean>true</boolean>
                    </field>
                    <field name="lifecyclePhase">
                        <collection type="java.util.ArrayList">
                            <value>
                                <string>read</string>
                            </value>
                        </collection>
                    </field>
                    <field name="roles">
                        <string>*:/platform/administrators</string>
                    </field>
                    <field name="variables">
                        <string>exo:to=benjamin.mestrallet@exoplatform.com</string>
                    </field>
                    <field name="mixins">
                        <collection type="java.util.ArrayList">
                            <value>
                                                                                <object
type="org.exoplatform.services.cms.actions.impl.ActionConfig$Mixin">
                                    <field name="name">
                                        <string>mix:affectedNodeTypes</string>
                                    </field>
                                    <field name="properties">
                                        <string>




exo:affectedNodeTypeNames=exo:article,exo:podcast,exo:sample,kfx:document,nt:file,rma:filePlan
                                        </string>
                                    </field>
```

```
                                </object>
                            </value>
                        </collection>
                    </field>
                </object>
            </value>
            <value>
            <object type="org.exoplatform.services.cms.actions.impl.ActionConfig$Action">
                <field name="type">
                    <string>exo:trashFolderAction</string>
                </field>
                <field name="name">
                    <string>trashFolder</string>
                </field>
                <field name="description">
                    <string>trigger actions for items in trash</string>
                </field>
                <field name="srcWorkspace">
                    <string>collaboration</string>
                </field>
                <field name="srcPath">
                    <string>/Trash</string>
                </field>
                <field name="isDeep">
                    <boolean>false</boolean>
                </field>
                <field name="lifecyclePhase">
                    <collection type="java.util.ArrayList">
                        <value>
                            <string>node_added</string>
                        </value>
                        <value>
                            <string>node_removed</string>
                        </value>
                    </collection>
                </field>
            </object>
        </value>
    </collection>
</field>
</object>
</object-param>
</init-params>
</component-plugin>
```

```
</external-component-plugins>
```

In which:

- **name**: `exo:scriptAction`
- **set-method** : `addPlugin`
- **type**: `org.exoplatform.services.cms.actions.impl.ScriptActionPlugin`
- **Object type**: `org.exoplatform.services.cms.actions.impl.ActionConfig`

Name	Type	Default Value	Description
repository	`string`	`repository`	The name of the repository.
workspace	`string`	`collaboration`	The name of the workspace.
actions	`ArrayList`	*{java.util.ArrayList}*	The list of the actions.

- **Object type**: `org.exoplatform.services.cms.actions.impl.ActionConfig$Action`

Name	Type	Default Value	Description
type	`string`	`exo:sendMailAction`	The type of the action.
name	`string`	`sendMail`	The name of the action.
description	`string`	`send a notification mail`	The description of the action.
srcWorkspace	`string`	`collaboration`	The source workspace of the action.
isDeep	`boolean`	`false`	Specify the depth of node that the action script will affect.
srcPath	`string`	`trash`	The path to the source.
lifecyclePhase	`ArrayList`	*{java.util.ArrayList}*	Specify the lifecycle phase that the action will take place.

ScriptPlugin

This plugin is used to add groovy scripts into the system.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.cms.scripts.ScriptService</target-component>
```

The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/dms-extension/dms/dms-scripts-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
   <target-component>org.exoplatform.services.cms.scripts.ScriptService</target-component>
   <component-plugin>
      <name>manage.script.plugin</name>
      <set-method>addScriptPlugin</set-method>
      <type>org.exoplatform.services.cms.scripts.impl.ScriptPlugin</type>
      <description>Nothing</description>
      <init-params>
         <value-param>
            <name>autoCreateInNewRepository</name>
            <value>true</value>
         </value-param>
         <value-param>
            <name>repository</name>
            <value>repository</value>
         </value-param>
         <value-param>
            <name>predefinedScriptsLocation</name>
            <value>war:/conf/dms-extension/dms/artifacts</value>
         </value-param>
         <object-param>
            <name>predefined.scripts</name>
            <description>description</description>
            <object type="org.exoplatform.services.cms.impl.ResourceConfig">
               <field name="resources">
                  <collection type="java.util.ArrayList">
                     <value>
                     <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                           <field name="name">
                              <string>ecm-explorer/action/RSSScript.groovy</string>
                           </field>
                        </object>
                     </value>
                     <value>
```

```
                    <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                        <field name="name">
                            <string>ecm-explorer/action/SendMailScript.groovy</string>
                        </field>
                    </object>
                </value>
                <value>
                <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                        <field name="name">
                            <string>ecm-explorer/action/TrashFolderScript.groovy</string>
                        </field>
                    </object>
                </value>
                <value>
                <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                        <field name="name">
                            <string>ecm-explorer/action/EnableVersioningScript.groovy</string>
                        </field>
                    </object>
                </value>
                <value>
                <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                        <field name="name">
                            <string>ecm-explorer/action/AutoVersioningScript.groovy</string>
                        </field>
                    </object>
                </value>
                <value>
                <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                        <field name="name">
                            <string>ecm-explorer/action/AddMetadataScript.groovy</string>
                        </field>
                    </object>
                </value>
                <value>
                <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                        <field name="name">
                        <string>ecm-explorer/action/TransformBinaryChildrenToTextScript.groovy</
string>
                        </field>
                    </object>
                </value>
                <value>
                <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
```

```
                        <field name="name">
                           <string>ecm-explorer/action/GetMailScript.groovy</string>
                        </field>
                     </object>
                  </value>
                  <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                        <field name="name">
                           <string>ecm-explorer/action/ProcessRecordsScript.groovy</string>
                        </field>
                     </object>
                  </value>
                  <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                        <field name="name">
                           <string>ecm-explorer/action/PublishingRequestScript.groovy</string>
                        </field>
                     </object>
                  </value>
                  <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                        <field name="name">
                           <string>ecm-explorer/action/AddTaxonomyActionScript.groovy</string>
                        </field>
                     </object>
                  </value>
                  <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                        <field name="name">
                     <string>ecm-explorer/widget/FillSelectBoxWithCalendarCategories.groovy</
string>
                        </field>
                     </object>
                  </value>
                  <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                        <field name="name">
                        <string>ecm-explorer/widget/FillSelectBoxWithMetadatas.groovy</string>
                        </field>
                     </object>
                  </value>
                  <value>
                  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
                        <field name="name">
```

```
        <string>ecm-explorer/widget/FillSelectBoxWithWorkspaces.groovy</string>
      </field>
    </object>
  </value>
  <value>
  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
      <field name="name">
    <string>ecm-explorer/widget/FillSelectBoxWithNodeChildren.groovy</string>
      </field>
    </object>
  </value>
  <value>
  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
      <field name="name">
       <string>ecm-explorer/widget/FillSelectBoxWithLanguage.groovy</string>
      </field>
    </object>
  </value>
  <value>
  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
      <field name="name">
     <string>ecm-explorer/interceptor/PreNodeSaveInterceptor.groovy</string>
      </field>
    </object>
  </value>
  <value>
  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
      <field name="name">
      <string>ecm-explorer/interceptor/PostNodeSaveInterceptor.groovy</string>
      </field>
    </object>
  </value>
  <value>
  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
      <field name="name">
       <string>ecm-explorer/interceptor/PostFilePlanInterceptor.groovy</string>
      </field>
    </object>
  </value>
  <value>
  <object type="org.exoplatform.services.cms.impl.ResourceConfig$Resource">
      <field name="name">
        <string>content-browser/GetDocuments.groovy</string>
      </field>
```

```
                        </object>
                    </value>
                </collection>
            </field>
        </object>
    </object-param>
  </init-params>
 </component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `manage.script.plugin`
- **Set-method**: `addScriptPlugin`
- **Type**: `org.exoplatform.services.cms.scripts.impl.ScriptPlugin`
- **Init-param**:

Value-param	Type	Value	Description
autoCreateInNewRepository	`boolean`	`true`	Enable/Disable the creation of the scripts in the newly created repository.
repository	`String`	`repository`	The repository name.
predefinedScriptsLocation	`String`	`war:/conf/dms-extension/dms/artifacts`	The location where the scripts are created.

- **Object type**: `org.exoplatform.services.cms.impl.ResourceConfig`

Field	Type	Value	Description
resource	`ArrayList`	*{java.util.ArrayList}*	The resource name.

StageAndVersionPublicationPlugin

This plugin is used to control the state life cycle of a content.

The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/content-extended/authoring/configuration.xml*.

Sample configuration:

```
<component-plugin>
```

```
   <name>States and versions based publication</name>
   <set-method>addPublicationPlugin</set-method>


type>
    <description>This publication lifecycle publish a web content or DMS document to a portal
 page with more state
      and version.
   </description>
</component-plugin>
```

In which:

- **name**: `States and versions based publication`
- **set method**: `addPublicationPlugin`
- **type**: `org.exoplatform.services.wcm.publication.lifecycle.stageversion.StageAndVersionPublicationPlu`

StatesLifecyclePlugin

This plugin is used to control the state life cycle of a content.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.wcm.extensions.publication.PublicationManager</
target-component>
```

The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/content-extended/authoring/configuration.xml*.

Sample configuration:

```
<component-plugin>
   <name>AddLifecycle</name>
   <set-method>addLifecycle</set-method>
    <type>org.exoplatform.services.wcm.extensions.publication.lifecycle.StatesLifecyclePlugin</
type>
   <description>Configures</description>
   <priority>1</priority>
   <init-params>
```

```
    <object-param>
      <name>lifecycles</name>
                                                                                <object
type="org.exoplatform.services.wcm.extensions.publication.lifecycle.impl.LifecyclesConfig">
        <field name="lifecycles">
          <collection type="java.util.ArrayList">
            <value>

                                                                                <object
type="org.exoplatform.services.wcm.extensions.publication.lifecycle.impl.LifecyclesConfig$Lifecycle">
                <field name="name">
                  <string>lifecycle1</string>
                </field>
                <field name="publicationPlugin">
                  <string>Authoring publication</string>
                </field>
                <field name="states">
                  <collection type="java.util.ArrayList">
                    <value>


                                                                                <object
type="org.exoplatform.services.wcm.extensions.publication.lifecycle.impl.LifecyclesConfig$State">
                          <field name="state">
                            <string>draft</string>
                          </field>
                          <field name="membership">
                            <string>author:/platform/web-contributors</string>
                          </field>
                        </object>
                    </value>
                    <value>


                                                                                <object
type="org.exoplatform.services.wcm.extensions.publication.lifecycle.impl.LifecyclesConfig$State">
                          <field name="state">
                            <string>pending</string>
                          </field>
                          <field name="membership">
                            <string>author:/platform/web-contributors</string>
                          </field>
                        </object>
                    </value>
```

```
                        <value>


                                                                                <object
type="org.exoplatform.services.wcm.extensions.publication.lifecycle.impl.LifecyclesConfig$State">
                            <field name="state">
                               <string>approved</string>
                            </field>
                            <field name="membership">
                               <string>manager:/platform/web-contributors</string>
                            </field>
                         </object>
                        </value>
                        <value>


                                                                                <object
type="org.exoplatform.services.wcm.extensions.publication.lifecycle.impl.LifecyclesConfig$State">
                            <field name="state">
                               <string>staged</string>
                            </field>
                            <field name="membership">
                               <string>publisher:/platform/web-contributors</string>
                            </field>
                         </object>
                        </value>
                        <value>


                                                                                <object
type="org.exoplatform.services.wcm.extensions.publication.lifecycle.impl.LifecyclesConfig$State">
                            <field name="state">
                               <string>published</string>
                            </field>
                            <field name="membership">
                               <string>publisher:/platform/web-contributors</string>
                            </field>
                         </object>
                        </value>
                     </collection>
                  </field>
               </object>
            </value>
         </collection>
```

```
            </field>
          </object>
        </object-param>
    </init-params>
</component-plugin>
```

In which:

- **Name**: `AddLifecycle`
- **Set-method**: `addLifecycle`
- **Type**: `org.exoplatform.services.wcm.extensions.publication.lifecycle.StatesLifecyclePlugin`
- **Object type**: `org.exoplatform.services.wcm.extensions.publication.lifecycle.impl.LifecyclesConfig$Lifecycle`

Field	Type	Value	Description
name	`String`	`lifecycle1`	The name of the lifecycle.
publicationPlugin	`string`	`Authoring publication`	The publication plugin name.
states	`ArrayList`	*{java.util.ArrayList}*	The list of the publication states.

- **Object type**: `org.exoplatform.services.wcm.extensions.publication.lifecycle.impl.LifecyclesConfig$State`

Field	Type	Description
state	`string`	The publication states: draft, pending, staged, approved or published.
membership	`string`	The user or group.

TagPermissionPlugin

This plugin is used to configure the predefined permission for tag to inject in JCR.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.cms.folksonomy.NewFolksonomyService</target-component>
```

The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/dms-extension/dms/dms-folksonomy-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
        <target-component>org.exoplatform.services.cms.folksonomy.NewFolksonomyService</target-component>
   <component-plugin>
      <name>predefinedTagPermissionPlugin</name>
      <set-method>addTagPermissionPlugin</set-method>
      <type>org.exoplatform.services.cms.folksonomy.impl.TagPermissionPlugin</type>
      <init-params>
         <object-param>
            <name>TagPermission.configuration</name>
            <description>configuration predefined permission for tag to inject in jcr</description>
            <object type="org.exoplatform.services.cms.folksonomy.impl.TagPermissionConfig">
               <field name="tagPermissionList">
                  <collection type="java.util.ArrayList">
                     <value>
                        <string>*:/platform/administrators</string>
                     </value>
                  </collection>
               </field>
            </object>
         </object-param>
      </init-params>
   </component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `predefinedTagPermissionPlugin`
- **Set-method**: `addTagPermissionPlugin`
- **Type**: `org.exoplatform.services.cms.folksonomy.impl.TagPermissionPlugin`
- **Object type**: `org.exoplatform.services.cms.folksonomy.impl.TagPermissionConfig`

Name	Type	Value	Description
tagPermissionList	`ArrayList`	*{java.util.ArrayList}*	The users/groups that have the permission.

TagStylePlugin

This plugin is used to configure the predefined styles for tag to inject in JCR.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.cms.folksonomy.NewFolksonomyService</target-component>
```

The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/dms-extension/dms/dms-folksonomy-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
        <target-component>org.exoplatform.services.cms.folksonomy.NewFolksonomyService</target-component>
   <component-plugin>
      <name>predefinedTagStylePlugin</name>
      <set-method>addTagStylePlugin</set-method>
      <type>org.exoplatform.services.cms.folksonomy.impl.TagStylePlugin</type>
      <init-params>
         <object-param>
            <name>htmStyleForTag.configuration</name>
            <description>configuration predefined html style for tag to inject in jcr</description>
            <object type="org.exoplatform.services.cms.folksonomy.impl.TagStyleConfig">
               <field name="autoCreatedInNewRepository">
                  <boolean>true</boolean>
               </field>
               <field name="repository">
                  <string>repository</string>
               </field>
               <field name="tagStyleList">
                  <collection type="java.util.ArrayList">
                     <value>
                                                                                    <object type="org.exoplatform.services.cms.folksonomy.impl.TagStyleConfig$HtmlTagStyle">
                           <field name="name">
                              <string>normal</string>
                           </field>
                           <field name="tagRate">
```

```
                        <string>0..2</string>
                      </field>
                      <field name="htmlStyle">
                        <string>font-size: 12px; font-weight: bold; color: #6b6b6b; font-family:
                          verdana; text-decoration:none;
                        </string>
                      </field>
                      <field name="description">
                        <string>Normal style for tag</string>
                      </field>
                    </object>
                  </value>
                  <value>
                                                                        <object
type="org.exoplatform.services.cms.folksonomy.impl.TagStyleConfig$HtmlTagStyle">
                      <field name="name">
                        <string>interesting</string>
                      </field>
                      <field name="tagRate">
                        <string>2..5</string>
                      </field>
                      <field name="htmlStyle">
                        <string>font-size: 13px; font-weight: bold; color: #5a66ce; font-family:
                          verdana; text-decoration:none;
                        </string>
                      </field>
                      <field name="description">
                        <string>Interesting style for tag</string>
                      </field>
                    </object>
                  </value>
                  <value>
                                                                        <object
type="org.exoplatform.services.cms.folksonomy.impl.TagStyleConfig$HtmlTagStyle">
                      <field name="name">
                        <string>attractive</string>
                      </field>
                      <field name="tagRate">
                        <string>5..7</string>
                      </field>
                      <field name="htmlStyle">
                        <string>font-size: 15px; font-weight: bold; color: blue; font-family: Arial;
                          text-decoration:none;
                        </string>
```

```
                        </field>
                        <field name="description">
                            <string>attractive style for tag</string>
                        </field>
                    </object>
                </value>
                <value>
                                                                                <object
type="org.exoplatform.services.cms.folksonomy.impl.TagStyleConfig$HtmlTagStyle">
                        <field name="name">
                            <string>hot</string>
                        </field>
                        <field name="tagRate">
                            <string>7..10</string>
                        </field>
                        <field name="htmlStyle">
                        <string>font-size: 18px; font-weight: bold; color: #ff9000; font-family: Arial;
                              text-decoration:none;
                            </string>
                        </field>
                        <field name="description">
                            <string>hot style for tag</string>
                        </field>
                    </object>
                </value>
                <value>
                                                                                <object
type="org.exoplatform.services.cms.folksonomy.impl.TagStyleConfig$HtmlTagStyle">
                        <field name="name">
                            <string>hottest</string>
                        </field>
                        <field name="tagRate">
                            <string>10..*</string>
                        </field>
                        <field name="htmlStyle">
                            <string>font-size: 20px; font-weight: bold; color: red; font-family:Arial;
                              text-decoration:none;
                            </string>
                        </field>
                        <field name="description">
                            <string>hottest style for tag</string>
                        </field>
                    </object>
                </value>
```

```
                    </collection>
                </field>
            </object>
        </object-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

- **Name**: `predefinedTagStylePlugin`
- **Set-method**: `addTagStylePlugin`
- **Type**: `org.exoplatform.services.cms.folksonomy.impl.TagStylePlugin`
- **Object type**: `org.exoplatform.services.cms.folksonomy.impl.TagStyleConfi`

Name	Type	Value	Description
autoCreatedInNewRepository	boolean	`true`	Specify whether the tag style is added automatically in a new reposistory or not.
repository	`string`	`repository`	Name of the repository where the tag style is added.
tagStyleList	`ArrayList`	*{java.util.ArrayList}*	The list of tag styles.

- **Object type**: `org.exoplatform.services.cms.folksonomy.impl.TagStyleConfig$HtmlTagStyle`

Name	Type	Description
name	`string`	The name of the tag.
tagRate	`string`	The number of times that a tag is used which will decide the respective tag style.
htmlStyle	`string`	The HTML code that defines the style.

TaxonomyPlugin

This plugin is used to configure the predefined taxonomies to inject into JCR.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.cms.taxonomy.TaxonomyService</target-component>
```

The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/dms-extension/dms/dms-categories-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
          <target-component>org.exoplatform.services.cms.taxonomy.TaxonomyService</target-component>
   <component-plugin>
      <name>predefinedTaxonomyPlugin</name>
      <set-method>addTaxonomyPlugin</set-method>
      <type>org.exoplatform.services.cms.taxonomy.impl.TaxonomyPlugin</type>
      <init-params>
         <value-param>
            <name>autoCreateInNewRepository</name>
            <value>true</value>
         </value-param>
         <value-param>
            <name>repository</name>
            <value>repository</value>
         </value-param>
         <value-param>
            <name>workspace</name>
            <value>dms-system</value>
         </value-param>
         <value-param>
            <name>treeName</name>
            <value>System</value>
         </value-param>
         <object-param>
            <name>permission.configuration</name>
            <object type="org.exoplatform.services.cms.taxonomy.impl.TaxonomyConfig">
               <field name="taxonomies">
                  <collection type="java.util.ArrayList">
                     <value>
                                                                        <object type="org.exoplatform.services.cms.taxonomy.impl.TaxonomyConfig$Taxonomy">
                           <field name="permissions">
```

```
                                <collection type="java.util.ArrayList">
                                    <value>
                                                                                                <object
type="org.exoplatform.services.cms.taxonomy.impl.TaxonomyConfig$Permission">
                                            <field name="identity">
                                                <string>*:/platform/users</string>
                                            </field>
                                            <field name="read">
                                                <string>true</string>
                                            </field>
                                            <field name="addNode">
                                                <string>true</string>
                                            </field>
                                            <field name="setProperty">
                                                <string>true</string>
                                            </field>
                                            <field name="remove">
                                                <string>false</string>
                                            </field>
                                        </object>
                                    </value>
                                </collection>
                            </field>
                        </object>
                    </value>
                </collection>
            </field>
        </object>
    </object-param>
    <object-param>
        ...
    </object-param>
  </init-params>
 </component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `predefinedTaxonomyPlugin`
- **Set-method**: `addTaxonomyPlugin`
- **Type**: `org.exoplatform.services.cms.taxonomy.impl.TaxonomyPlugin`
- **Init-param**:

Value-param	Type	Value	Description
autoCreateInNewRepository	`Boolean`	`true`	Enable/Disable the creation of the taxonomies in the newly created repository.
repository	`String`	`repository`	The name of the repository where taxonomies are created.
workspace	`String`	`dms-system`	The name of the workspace where taxonomies are created.
treeName	`String`	`system`	The name of the taxonomy tree created.

- **Object type**: `org.exoplatform.services.cms.taxonomy.impl.TaxonomyConfig`

Name	Type	Value	Description
taxonomies	`ArrayList`	*{java.util.ArrayList}*	The list of taxonomies to be configured with permission.

- **Object type**: `org.exoplatform.services.cms.taxonomy.impl.TaxonomyConfig$Taxonomy`

Name	Type	Value	Description
permissions	`ArrayList`	*{java.util.ArrayList}*	The list of permissions for users or groups to access the taxonomy.

- **Object type**: `org.exoplatform.services.cms.taxonomy.impl.TaxonomyConfig$Permission`

Name	Type	Value	Description
identity	`String`	*:/platform/users	The name of the user, group or membership.
read	`Boolean`	`true`	The permission to read the taxonomy tree.
addNode	`Boolean`	`true`	The permission to add a node to the taxonomy tree.

Name	Type	Value	Description
setProperty	`Boolean`	`true`	The permission to set properties for a node in the taxonomy tree.
remove	`Boolean`	`false`	The permission to remove a node from the taxonomy tree.

TemplatePlugin

This plugin is used to create templates into the system. A template is a presentation to display the saved information.

The node type template is used to edit and display the node content. Each node type has one *dialog1.gtmpl* file (dialog template) for editing/creating a node and one *view1.gtmpl* file (view template) for viewing the node content. Using the dialog template, you can specify a dialog whose fields correspond to the properties of the node you want to edit their values. When this template is rendered, each specified field will appear with a data input box for you to edit. Note that you do not have to design a dialog in which all data of the node are listed to be edited. You can just list the subset of node data you want to edit. Like the dialog template, the view template renders information of the node. You just need to create the template and specify which data fields to be displayed. With this kind of template, node information is only displayed but cannot be edited. See details at *ContentType*.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.cms.templates.TemplateService</target-component>
```

The configuration is applied mainly in *packaging/wcm/webapp/src/main/webapp/WEB-INF/conf/dms-extension/dms/dms-templates-configuration.xml*.

Sample configuration:

This below example is configuration for the *nt:file* template, any other template will be put in the same level with this template starting from the line `<object type="org.exoplatform.services.cms.templates.impl.TemplateConfig$NodeType">` as the another node type.

```
<external-component-plugins>
        <target-component>org.exoplatform.services.cms.templates.TemplateService</target-component>
```

```
  <component-plugin>
    <name>addTemplates</name>
    <set-method>addTemplates</set-method>
    <type>org.exoplatform.services.cms.templates.impl.TemplatePlugin</type>
    <init-params>
      <value-param>
        <name>autoCreateInNewRepository</name>
        <value>true</value>
      </value-param>
      <value-param>
        <name>storedLocation</name>
        <value>war:/conf/dms-extension/dms/artifacts/templates</value>
      </value-param>
      <value-param>
        <name>repository</name>
        <value>repository</value>
      </value-param>
      <object-param>
        <name>template.configuration</name>
        <description>configuration for the localtion of templates to inject in jcr</description>
        <object type="org.exoplatform.services.cms.templates.impl.TemplateConfig">
          <field name="nodeTypes">
            <collection type="java.util.ArrayList">
              <value>
                                                                    <object
type="org.exoplatform.services.cms.templates.impl.TemplateConfig$NodeType">
                    <field name="nodetypeName">
                      <string>nt:file</string>
                    </field>
                    <field name="documentTemplate">
                      <boolean>true</boolean>
                    </field>
                    <field name="label">
                      <string>File</string>
                    </field>
                    <field name="referencedView">
                      <collection type="java.util.ArrayList">
                        <value>
                                                                    <object
type="org.exoplatform.services.cms.templates.impl.TemplateConfig$Template">
                              <field name="templateFile">
                                <string>/file/views/view1.gtmpl</string>
                              </field>
                              <field name="roles">
```

```
                                    <string>*</string>
                                </field>
                            </object>
                        </value>
                        <value>
                                                                                <object
type="org.exoplatform.services.cms.templates.impl.TemplateConfig$Template">
                                <field name="templateFile">
                                    <string>/file/views/admin_view.gtmpl</string>
                                </field>
                                <field name="roles">
                                    <string>*:/platform/administrators</string>
                                </field>
                            </object>
                        </value>
                    </collection>
                </field>
                <field name="referencedDialog">
                    <collection type="java.util.ArrayList">
                        <value>
                                                                                <object
type="org.exoplatform.services.cms.templates.impl.TemplateConfig$Template">
                                <field name="templateFile">
                                    <string>/file/dialogs/dialog1.gtmpl</string>
                                </field>
                                <field name="roles">
                                    <string>*</string>
                                </field>
                            </object>
                        </value>
                        <value>
                                                                                <object
type="org.exoplatform.services.cms.templates.impl.TemplateConfig$Template">
                                <field name="templateFile">
                                    <string>/file/dialogs/admin_dialog.gtmpl</string>
                                </field>
                                <field name="roles">
                                    <string>*:/platform/administrators</string>
                                </field>
                            </object>
                        </value>
                    </collection>
                </field>
                <field name="referencedSkin">
```

```
                        <collection type="java.util.ArrayList">
                            <value>
                                                                                <object
type="org.exoplatform.services.cms.templates.impl.TemplateConfig$Template">
                                    <field name="templateFile">
                                        <string>/file/skins/Stylesheet-lt.css</string>
                                    </field>
                                    <field name="roles">
                                        <string>*</string>
                                    </field>
                                </object>
                            </value>
                            <value>
                                                                                <object
type="org.exoplatform.services.cms.templates.impl.TemplateConfig$Template">
                                    <field name="templateFile">
                                        <string>/file/skins/Stylesheet-rt.css</string>
                                    </field>
                                    <field name="roles">
                                        <string>*</string>
                                    </field>
                                </object>
                            </value>
                        </collection>
                    </field>
                </object>
            </value>
        </collection>
    </field>
  </object>
 </object-param>
 </init-params>
 </component-plugin>
</external-component-plugins>
```

In which:

- **name**: `addTemplates`
- **set-method**: `addTemplates`
- **type**: `org.exoplatform.services.cms.templates.impl.TemplatePlugin`
- **Init-params**:

Value-param	Type	Value	Description
autoCreateInNewRepository	`Boolean`	`true`	Enable the application to import predefined templates at the start-up of template service automatically.
storedLocation	`String`	`war:/conf/dms-extension/dms/artifacts/templates`	The location of stored templates.
repository	`String`	`repository`	Location of stored templates.

- **Object-type**: `org.exoplatform.services.cms.templates.impl.TemplateConfig` that defines all available template files, using the "collection type" configuration.

- **type**: It is the name of each object type. It means the type of template, the further configurations for this type are defined by some specified fields.

Field	Type	Value	Description
nodeTypes	`ArrayList`	{java.util.ArrayList}	The node type of the template.

- **Object-type**: `org.exoplatform.services.cms.templates.impl.TemplateConfig$NodeType`

Field	Type	Value	Description
nodetypeName	`String`	nt:file	The name of template that is saved as a node in system.
documentTemplate	`Boolean`	`true`	Determine if the node type is a document type.
label	`String`	`file`	Visual display of the title for this node.
referencedView	`ArrayList`	{java.util.ArrayList}	Determine how to display a view.
referencedDialog	`ArrayList`	{java.util.ArrayList}	Determine how to display a dialog to input information.

Field	Type	Value	Description
referencedSkin	`ArrayList`	{java.util.ArrayList}	Determine the stylesheet for display.

- **Object type:** `org.exoplatform.services.cms.templates.impl.TemplateConfig$Template`

Field	Type	Description
templateFile	`String`	The location of the file store for the template's presentation.
roles	`String`	Determine who can access this object (View/Dialog/CSS).

XMLdeploymentPlugin

When a site is created, most of end-users want to see something in the page instead of a blank page, so you need this plugin to deploy some "default" contents, such as Banner, Footer, Navigation, Breadcrumb.

There are two main cases to use:

- The site is created only one time when the database is cleaned.
- The site is created at runtime, when a user uses the core features of the GateIn portal.

To use the plugin in the component configuration, you must use the following target-component:

```
<target-component>org.exoplatform.services.deployment.WCMContentInitializerService</
target-component>
```

The configuration is applied mainly in *packaging/ecmdemo/webapp/src/main/webapp/WEB-INF/conf/sample-portal/wcm/deployment/acme-deployment-configuration.xml*.

Sample configuration:

```
<external-component-plugins>
        <target-component>org.exoplatform.services.deployment.WCMContentInitializerService</
target-component>
    <component-plugin>
```

```
      <name>Content Initializer Service</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.deployment.plugins.XMLDeploymentPlugin</type>
      <description>XML Deployment Plugin</description>
      <init-params>
        <object-param>
          <name>ACME Logo data</name>
          <description>Deployment Descriptor</description>
          <object type="org.exoplatform.services.deployment.DeploymentDescriptor">
            <field name="target">
            <object type="org.exoplatform.services.deployment.DeploymentDescriptor$Target">
                <field name="repository">
                  <string>repository</string>
                </field>
                <field name="workspace">
                  <string>collaboration</string>
                </field>
                <field name="nodePath">
                  <string>/sites content/live/acme/web contents/site artifacts</string>
                </field>
              </object>
            </field>
            <field name="sourcePath">
           <string>war:/conf/sample-portal/wcm/artifacts/site-resources/acme/Logo.xml</string>
            </field>
            <field name="versionHistoryPath">
                              <string>war:/conf/sample-portal/wcm/artifacts/site-resources/acme/
Logo_versionHistory.zip
              </string>
            </field>
            <field name="cleanupPublication">
              <boolean>true</boolean>
            </field>
          </object>
        </object-param>
      </init-params>
    </component-plugin>
</external-component-plugins>
```

In which:

- **name**: `Content Initializer Service`
- **set-method**: `addPlugin`

- **type**: `org.exoplatform.services.deployment.plugins.XMLDeploymentPlugin`

- **Object type**: `org.exoplatform.services.deployment.DeploymentDescriptor`

Name	Type	Value	Description
target	`Object`	`org.exoplatform.services.deployment.DeploymentDescriptor$Target (*)`	The target node which will contain the imported node.
sourcePath	`String`	`war:/conf/sample-portal/wcm/artifacts/site-resources/acme/Logo.xml<`	The absolute path of the XML file.
cleanupPublication	`Boolean`	`false`	Decide when the publication lifecycle is cleaned up in the target folder after importing the data. true: allow false:not allow
versionHistoryPath	`String`	`war:/conf/sample-portal/wcm/artifacts/site-resources/acme/Logo_versionHistory.zip<`	The absolute path of the version history file.

- **Object type**: `org.exoplatform.services.deployment.DeploymentDescriptor$Target`

Field	Type	Value	Description
repository	`String`	`repository`	The repository name of the target node.
workspace	`String`	`collaboration`	The collaboration name of the target node.
nodePath	`String`	`/sites content/live/acme/web contents/site artifacts`	The path of the target node.

Developer references

This chapter supply you with the basic knowledge about templates, UI extension, APIs in eXo Content. Through it, you can build your own content. Also, with the step-by-step guides, you can create UI extension, deploy a workflow in eXo Content and do many different actions.

This chapter consists of the main following contents:

WCM Templates

Content types

Overview

The templates are applied to a node type or a metadata mixin type. There are two types of templates:

- **Dialogs**: are in the HTML form that allows creating node instances.
- **Views**: are in the HTML fragments which are used to display nodes.

From the ECM admin portlet, the *Manage Template* lists existing node types associated to Dialog and/or View templates. These templates can be attached to permissions (in the usual *membership:group* form), so that a specific one is displayed according to the rights of the user (very useful in a content validation workflow activity).

Document Type

The checkbox defines if the node type should be considered as the **Document type** or not. Content Explorer considers such nodes as user content and applies the following behavior:

- View template will be used to display the document type nodes.
- Document types nodes can be created by the '*Add Document*' action.
- Non-document types are hidden (unless the *'Show non document types'* option is checked).

Templates are written by using *Groovy Templates* [http://groovy.codehaus.org/Groovy+Templates] that requires some experiences with JCR API and HTML notions.

Dialog

Dialogs are Groovy templates that generate forms by mixing static HTML fragments and Groovy calls to the components responsible for building the UI at runtime. The result is a simple but powerful syntax.

Common parameters

These following parameters are common and can be used for all input fields.

Parameter	Type	Required	Description	Example
jcrPath	`String`		The relative path inside the current node.	`jcrPath=/node/exo:title`
mixintype	`String` with the commas (,) character.		The list of mixin types you want to initialize when creating the content.	`mixintype=mix:i18n` `mixintype=mix:votable,mix:comme`

validate	`String` with the comma (,) character		The list of validators you want to apply to the input. Possible values are: *name, email, number, empty, null, datetime, length* OR validator classes.	`validate=empty`
				`validate=empty,name`
				`validate=org.exoplatform.webui.`

editable	`String`		The input will be editable only if the value of this parameter is `if-null` and the value of this input is null or blank.	`editable=if-null`
multiValues	`Boolean`		Show a multi-valued component if true and must be used only with	`multiValues=true`

Parameter	Type	Required	Description	Example
			corresponding multi-valued properties. The default value of this parameter is false.	
visible	`Boolean`		The input is visible if this value is true.	`visible=true`

See also:

- *Text Field*
- *Hidden Field*
- *Text Area Field*
- *Rich Text Field*
- *Calendar Field*
- *Upload Field*
- *Radio Field*
- *Select Box Field*
- *Checkbox Field*
- *Mixin Field*
- *Action Field*

Note

The *mixintype* can be used only in the root node field (commonly known as the name field).

Text Field

- **Additional parameters**

See also: *Common parameters*

- **Example**

```
<%
  String[] fieldTitle = ["jcrPath=/node/exo:title", "validate=empty"] ;
  uicomponent.addTextField("title", fieldTitle) ;
%>
```

Hidden Field

- **Additional parameters**

See also: *Common parameters*

- **Example**

```
String[] hiddenField5 = ["jcrPath=/node/jcr:content/dc:date", "visible=false"];
  uicomponent.addCalendarField("hiddenInput5", hiddenField5);
```

Text Area Field

- **Additional parameters**

Parameter	Type	Required	Description	Example
rows	`Number`		The initial text area's number of rows. The value is `10` by default.	`rows=20`
cols	`Number`		The initial text area's number of cols. The value is `30` by default .	`cols=50`

See also: *Common parameters*

- **Example**

```
<%
  String[] fieldDescription = ["jcrPath=/node/exo:description", "validate=empty"] ;
  uicomponent.addTextAreaField("description", fieldDescription) ;
%>
```

Rich Text Field

- **Additional parameters**

Parameter	Type	Required	Description	Example
options	`String` with the semicolon (;) character	✗	Some options for CKEditor field: toolbar, width and height.	`options=CompleteWCM;width:'100%`
toolbar	`String`	✗	The predefined toolbar for CKEditor. The value can be: `Default`, `Basic`, `CompleteWCM`, `BasicWCM`, `SuperBasicWCM`.	`options=CompleteWCM`
width	`String`	✗	The width of CKEditor. Its value can be the percent of pixel.	`options=width:'100%'`
height	`String`	✗	The height of CKEditor. Its value can be the percent of pixel.	`options=height:'200px'`

See also: *Common parameters*

- **Example**

```
<%
                String[]  fieldSummary  =  ["jcrPath=/node/exo:summary",
 "options=toolbar:CompleteWCM,width:'100%',height:'200px'", "validate=empty"] ;
  uicomponent.addRichtextField("summary", fieldSummary) ;
%>
```

Calendar Field

- **Additional parameters**

Parameter	Type	Required	Description	Example
options	`String`		An option for the calendar field: Display time.	`options=displaytime`

See also: *Common parameters*

- **Example**

```
<%
    String[] fieldPublishedDate = ["jcrPath=/node/exo:publishedDate", "options=displaytime",
 "validate=datetime", "visible=true"] ;
  uicomponent.addCalendarField("publishedDate", fieldPublishedDate) ;
%>
```

Upload Field

- **Additional parameters**

See also: *Common parameters*

- **Example**

When you create an upload form, you can store an image by two main ways:

- If you want to store the image as a property, use the following code:

```
<%
  String[] fieldMedia = ["jcrPath=/node/exo:image"] ;
  uicomponent.addUploadField("media", fieldMedia) ;
%>
```

- If you want to store the image as a node, use the following code:

```
<%
  String[] hiddenField1 = ["jcrPath=/node/exo:image", "nodetype=nt:resource", "visible=false"] ;
```

```
  String[] hiddenField2 = ["jcrPath=/node/exo:image/jcr:encoding", "visible=false", "UTF-8"] ;
  String[] hiddenField3 = ["jcrPath=/node/exo:image/jcr:lastModified", "visible=false"] ;
  uicomponent.addHiddenField("hiddenInput1", hiddenField1) ;
  uicomponent.addHiddenField("hiddenInput2", hiddenField2) ;
  uicomponent.addHiddenField("hiddenInput3", hiddenField3) ;

  String[] fieldMedia = ["jcrPath=/node/exo:image"] ;
  uicomponent.addUploadField("media", fieldMedia) ;
%>
```

- But, this code is not complete. If you want to display the **upload** field, the image must be blank, otherwise you can display the image and an action enables you to remove it. You can do as follows:

```
<%
 def image = "image";
 // If you're trying to edit the document
 if(uicomponent.isEditing()) {
  def curNode = uicomponent.getNode();
  // If the image existed
  if (curNode.hasNode("exo:image")) {
   def imageNode = curNode.getNode("exo:image") ;
   // If the image existed and available
                  if      (imageNode.getProperty("jcr:data").getStream().available()      >      0      &&
 (uicomponent.findComponentById(image) == null)) {
    def imgSrc = uicomponent.getImage(curNode, "exo:image");
    def actionLink = uicomponent.event("RemoveData", "/exo:image");
    %>
     <div>
      <img src="$imgSrc" width="100px" height="80px"/>
      <a href="$actionLink">
        <img src="/eXoResources/skin/DefaultSkin/background/Blank.gif" alt="" class="ActionIcon
 Remove16x16Icon"/>
      </a>
     </div>
    <%
   } else {
    String[] fieldImage = ["jcrPath=/node/exo:image/jcr:data"] ;
    uicomponent.addUploadField(image, fieldImage) ;
   }
  } else {
    String[] fieldImage = ["jcrPath=/node/exo:image/jcr:data"] ;
```

```
    uicomponent.addUploadField(image, fieldImage) ;
  }
 }  else if(uicomponent.dataRemoved()) {
  String[] fieldImage = ["jcrPath=/node/exo:image/jcr:data"] ;
  uicomponent.addUploadField(image, fieldImage) ;
 } else {
  String[] fieldImage = ["jcrPath=/node/exo:image/jcr:data"] ;
  uicomponent.addUploadField(image, fieldImage) ;
 }
%>
```

Radio Field

- **Additional parameters**

Parameter	Type	Required	Description	Example
options	`String` with the comma (,) characters		Some radio values.	`options=radio1,radio2,radio3`

See also: *Common parameters*

- **Example**

```
<%
        String[] fieldDeep = ["jcrPath=/node/exo:isDeep", "defaultValues=true",
 "options=radio1,radio2,radio3"];
  uicomponent.addRadioBoxField("isDeep", fieldDeep);
%>
```

Select box Field

Parameter	Type	Required	Description	Example
options	`String` with the comma (,) characters		Some option values.	`options=option1,option2,option3`

See also: *Common parameters*

- **Example**

```
<%
        String[]  fieldDeep  =  ["jcrPath=/node/exo:isDeep",  "defaultValues=true",
 "options=checkbox1,checkbox2,checkbox3"];
  uicomponent.addCheckBoxField("isDeep", fieldDeep);
%>
```

Checkbox Field

- **Additional parameters**

Parameter	Type	Required	Description	Example
options	`String` with the comma (,) characters		Some checkbox values.	`options=checkbox1,checkbox2,che`

See also: *Common parameters*

- **Example**

```
<%
        String[]  fieldDeep  =  ["jcrPath=/node/exo:isDeep",  "defaultValues=true",
 "options=checkbox1,checkbox2,checkbox3"];
  uicomponent.addCheckBoxField("isDeep", fieldDeep);
%>
```

Mixin Field

- **Additional parameters**

See also: *Common parameters*

- **Example**

```
<%
  String[] fieldId = ["jcrPath=/node", "editable=false", "visible=if-not-null"] ;
  uicomponent.addMixinField("id", fieldId) ;
```

```
%>
```

Action Field

- **Additional parameters**

Parameter	Type	Required	Description	Example
selectorClass	`String`	✓	The component to display.	`selectorClass=org.exoplatform.e`
selectorIcon	`String`	✓	The action icon.	`selectorIcon=SelectPath24x24Ico`

Depending on the `selectorClass`, some other parameters can be added.

For example, the component `org.exoplatform.ecm.webui.tree.selectone.UIOneNodePathSelector` needs the following parameter:

Parameter	Type	Required	Description	Example
workspaceField	`String`	✗	The field which enables you to select a workspace.	`workspaceField=targetWorkspace`

The component `org.exoplatform.ecm.webui.selector.UIPermissionSelector` does not need any special parameters.

See also: *Common parameters*

- **Example**

```
<%
                String[]  fieldPath  =  ["jcrPath=/node/exo:targetPath",

 "workspaceField=targetWorkspace", "selectorIcon=SelectPath24x24Icon"] ;
  uicomponent.addActionField("targetPath", fieldPath)  ;
%>
```

Interceptors

To add an interceptor to a dialog, you can use this method **`uicomponent.addInterceptor(String scriptPath, String type)`**

Parameters	Type	Description
scriptPath	`String`	The relative path to the script file.
type	`String`	The type of interceptor: `prev` or `post`.

- **Example**

```
<%
        uicomponent.addInterceptor("ecm-explorer/interceptor/PreNodeSaveInterceptor.groovy",
 "prev");
%>
```

How to add a new ECM template with tabs

To avoid refreshing the first tab for every action execution, add a new private function to the template with tabs. In the template, you must insert a new piece of code like the following:

```
private String getDisplayTab(String selectedTab) {
  if ((uicomponent.getSelectedTab() == null && selectedTab.equals("mainWebcontent"))
    || selectedTab.equals(uicomponent.getSelectedTab())) {
  return "display:block";
 }
 return "display:none";
}

private String getSelectedTab(String selectedTab) {
 if (getDisplayTab(selectedTab).equals("display:block")) {
  return "SelectedTab";
 }
 return "NormalTab";
}
```

Changing in every event of **onClick** must be done like the following:

```
<div class="UITab NormalTabStyle">
  <div class="<%=getSelectedTab("mainWebcontent")%>
    ">
```

```
   <div class="LeftTab">
    <div class="RightTab">
                        <div    class="MiddleTab"    onClick="<%=uicomponent.event("ChangeTab",
 "mainWebcontent")%>"><%=_ctx.appRes("WebContent.dialog.label.MainContent")%></div>
    </div>
   </div>
  </div>
</div>

<div class="UITab NormalTabStyle">
  <div class="<%=getSelectedTab("illustrationWebcontent")%>
   ">
   <div class="LeftTab">
    <div class="RightTab">
                        <div    class="MiddleTab"    onClick="<%=uicomponent.event("ChangeTab",
 "illustrationWebcontent")%>"><%=_ctx.appRes("WebContent.dialog.label.Illustration")%></div>
    </div>
   </div>
  </div>
</div>

<div class="UITab NormalTabStyle">
  <div class="<%= getSelectedTab("contentCSSWebcontent")%>
   ">
   <div class="LeftTab">
    <div class="RightTab">
                        <div    class="MiddleTab"    onClick="<%=uicomponent.event("ChangeTab",

div>
    </div>
   </div>
  </div>
</div>
```

Finally, to display the selected tab, simply add it to the style of UITabContent class.

```
<div class="UITabContent" style="<%=getDisplayTab("mainWebcontent")%>">
```

View

The following is an sample code of the **View** template of a content node:

- Get a content node to display:

```
<%
  def node = uicomponent.getNode() ;
  def originalNode = uicomponent.getOriginalNode()
%>
```

- Display the name of the content node:

```
<%=node.getName()%>
```

- Display the *exo:title* property of the content node:

```
<%
  if(node.hasProperty("exo:title")) {
%>
  <%=node.getProperty("exo:title").getString()%>
<%
  }
%>
```

- Display the *exo:date* property of the content node in a desired format. For example: "MM DD YYYY" or "YYYY MM DD".

```
<%
  import java.text.SimpleDateFormat ;
  SimpleDateFormat dateFormat = new SimpleDateFormat() ;
%>
...

<%
  if(node.hasProperty("exo:date")) {
    dateFormat.applyPattern("MMMMM dd yyyy") ;
%>
```

```
   <%=dateFormat.format(node.getProperty("exo:date").getDate().getTime())%>
<%
 }
%>
```

- Display the translation of the 'Sample.view.label.node-name' message in different lanaguages.

```
<%=_ctx.appRes("Sample.view.label.node-name")%>
```

List of Contents

Content List Template

The **Content List Template** allows you to view the content list with various templates. eXo Platform supports the following content list templates:

Template	Description
BigHotNewsTemplateCLV.gtmpl	Display contents under one column with a content list. The illustration of each content is displayed above the content.
ContentListViewerDefault.gtmpl	Its function is similar to **BigHotNewsTemplateCLV.gtmpl**. The illustration of each content is bigger.
DocumentsTemplate.gtmpl	Display contents under a content list with a NodeType icon or the illustration on the left of the corresponding content.
EventsTemplateCLV.gtmpl	Its function is similar to **BigHotNewsTemplateCLV.gtmpl**, but the illustration of each content is smaller.
OneColumnCLVTemplate.gtmpl	Display contents under one column. The illustration of each content is displayed on its left.
TwoColumnsCLVTemplate.gtmpl	Display contents under two columns. The illustration of each content is displayed on its left.
UIContentListPresentationBigImage.gtmpl	Its function is similar to **BigHotNewsTemplateCLV.gtmpl**, but the illustration of each content is bigger

Template	Description
	than the image displayed with **ContentListViewerDefault.gtmpl** and the text font is different.
UIContentListPresentationDefault.gtmpl	Its function is similar to **BigHotNewsTemplateCLV.gtmpl**, but the illustration of each content is smaller and the text font is different.
UIContentListPresentationSmall.gtmpl	Display contents under one column with a content list. The images are displayed on the left of the corresponding content and smaller than the images of the other templates.

Category Navigation Template

The **Category Navigation Template** displays all contents under the categories.

Template	Description
CategoryList.gtmpl	Display categories as a navigation bar.
CategoryTree.gtmpl	Display categories as a tree.
TagsCloud.gtmpl	Display all tags of the contents.

WCM Explorer

CSS

- By using WCM, all the stylesheets of each site can be managed online easily. You do not need to access the file system to modify and wait until the server has been restarted. For the structure, each site has its own CSS folder which can contain one or more CSS files. These CSS files have the data, and the priority. If they have the same CSS definition, the higher priority will be applied. You can also disable some of them to make sure the disabled style will no longer be applied into the site.

- For example, a WCM demo package has two sites by default: ACME and Classic. The Classic site has a CSS folder which contains a CSS file called **DefaultStylesheet**. Most of the stylesheets of this site are defined within this stylesheet. Moreover, the ACME site has two CSS files called **BlueStylesheet** and **GreenStylesheet**. The blue one is enabled and the green one is disabled by default. All you need to test is to disable the blue one (by editing it and setting Available equals false) and enable the green one. Now, back to the homepage and see the magic.

Note

Remember the cache and refresh the browser first if you do not see any changes. Normally, this is the main reason why the new style is not applied.

CKEditor

Basically, if you want to add a rich text area to your dialogs, you can use the *addRichtextField* method. However, in case you want to add the rich text editor manually, you first need to use the *addTextAreaField* method and some additional Javascripts as shown below:

```
<%
 String[] fieldDescription = ["jcrPath=/node/exo:description"] ;
 uicomponent.addTextAreaField("description", fieldDescription)
%>
<script>
  var instances = CKEDITOR.instances['description'];
  if (instances) instances.destroy(true);
  CKEDITOR.replace('description', {
    toolbar : 'CompleteWCM',
    uiColor : '#9AB8F3'
  });
</script>
```

Extensions

REST Services

Overview

REST-style architectures consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of "representations" of "resources". A resource can be essentially any coherent and meaningful concept that may be addressed. A representation of a resource is typically a document that captures the current or intended state of a resource.

At any particular time, a client can either be in transition between application states or "at rest". A client in a REST state is able to interact with its users, but creates no load and consumes no per-client storage on the set of servers or on the network.

The client begins sending requests when it is ready to make the transition to a new state. While one or more requests are outstanding, the client is considered to be in transition. The representation of each application state contains links that may be used the next time, the client chooses to initiate a new state transition.

REST is initially described in the context of HTTP, but is not limited to that protocol. RESTful architectures can be based on other Application Layer protocols if they already provide a rich and uniform vocabulary for applications based on the transfer of meaningful representational state. RESTful applications maximize the use of the pre-existing, well-defined interface and other built-in capabilities provided by the chosen network protocol, and minimize the addition of new application-specific features on its top.

Restful Web Service

HTTP Methods

Here is the convention you should follow:

Method	Definition
GET	Get a Resource. Its state should not be modified.
POST	Create a Resource (or anything that does not fit elsewhere).
PUT	Update a Resource.
DELETE	Delete a Resource.

Formats

The followings are formats which need to be supported for all your APIs:

- *JSON* [http://tools.ietf.org/html/rfc4627]: This format makes developers easy to parse in a lot of languages, such as JavaScript, Python or Ruby.
- *XML* [http://www.w3.org/TR/xml/]: Most of Java developers like using this format.
- *ATOM* [http://tools.ietf.org/html/rfc4287]: This is a standard format which can be used by many applications.

Data Format

The default format is JSON.

The response format can be specified by a parameter in the request: "*format*". You need to specify the format requested.

REST configuration

First, you need to register the REST service class to the configuration file in the package named **conf.portal** .

```
<configuration xmlns="http://www.exoplaform.org/xml/ns/kernel_1_1.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_1.xsd http://www.exoplaform.org/xml/ns/kernel_1_1.xsd">
  <component>


type>
  </component>
</configuration>
```

Create a REST service

You can start creating GetEditedDocumentRESTService that implements from the ResourceContainer interface as follows:

```
@Path("/presentation/document/edit/")
public class GetEditedDocumentRESTService implements ResourceContainer {
 @Path("/{repository}/")
  @GET
  public Response getLastEditedDoc(@PathParam("repository") String repository,
     @QueryParam("showItems") String showItems) throws Exception {
     ........
  }
}
```

Parameters	Definition
@Path("/presentation/document/edit/")	Specify the URI path which a resource or class method will serve requests for.
@PathParam("repository")	Bind the value **repository** of a URI parameter or a path segment containing the template parameter to a resource method parameter, resource class field, or resource class bean property.
@QueryParam("showItems")	

Parameters	Definition
	Bind the value **showItems** of a HTTP query parameter to a resource method parameter, resource class field, or resource class bean property.

UI Extensions

This part describes how to create a sample UI extension.

Overview

In eXo Content, it is possible to extend the **Content Explorer** and the **ECM Administration** with the **UI Extension Framework**. Indeed, you can add your own action buttons to the **Content Explorer** and/or add your own managers to the **ECM Administration**.

How to add your own tab in ECM Administration

Add your own UIAction

To add your own UIAction, do as follows:

1. Create a *pom.xml* file with the following content:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.exoplatform.ecms</groupId>
    <artifactId>exo-ecms-examples-uiextension-framework</artifactId>
    <version>2.2.0-SNAPSHOT</version>
  </parent>
  <artifactId>exo-ecms-examples-uiextension-framework-manage-wcm-cache</artifactId>
  <name>eXo WCM Cache Examples</name>
  <description>eXo WCM Cache Examples</description>
  <dependencies>
    <dependency>
      <groupId>org.exoplatform.kernel</groupId>
      <artifactId>exo.kernel.container</artifactId>
      <version>2.3.2-GA</version>
      <scope>compile</scope>
    </dependency>
```

```
    <dependency>
      <groupId>org.exoplatform.commons</groupId>
      <artifactId>exo.platform.commons.webui.ext</artifactId>
      <version>1.1.3</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>org.exoplatform.ecms</groupId>
      <artifactId>exo-ecms-core-webui</artifactId>
      <version>2.3.2</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>org.exoplatform.ecms</groupId>
      <artifactId>exo-ecms-core-webui-administration</artifactId>
      <version>2.3.2</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
  <build>
    <resources>
      <resource>
        <directory>src/main/java</directory>
        <includes>
          <include>**/*.xml</include>
        </includes>
      </resource>
      <resource>
        <directory>src/main/resources</directory>
        <includes>
          <include>**/*.properties</include>
          <include>**/*.xml</include>
          <include>**/*.jar</include>
          <include>**/*.pom</include>
          <include>**/*.conf</include>
          <include>**/*.gtmpl</include>
          <include>**/*.gif</include>
          <include>**/*.jpg</include>
          <include>**/*.png</include>
        </includes>
      </resource>
    </resources>
  </build>
```

```
</project>
```

2. Create the *src/main/java* directory and start launching *mvn eclipse:eclipse*. Then, you can launch your eclipse and import this new project.

3. Create a new class called *org.exoplatform.wcm.component.cache.UIWCMCacheComponent* that extends *org.exoplatform.ecm.webui.component.admin.manager.UIAbstractManagerComponent*.

Add your own ActionListener

With the Webui framework, you will be notified if any given action is triggered. You just need to call your own action listener ($ACTIONNAME) **ActionListener**. For example, to create your own action listener for **CacheView**, do as follows:

1. Call **CacheViewActionListener**.

2. Add a static inner class called *CacheViewActionListener* that extends *org.exoplatform.ecm.webui.component.admin.listener.UIECMAdminControlPanelActionListener*.

You will see the expected code below:

```
public class CacheViewComponent extends UIAbstractManagerComponent {
                                public static class CacheViewActionListener extends
 UIECMAdminControlPanelActionListener
<UIWCMCacheComponent>{
   public void processEvent(Event
   <UIWCMCacheComponent>event) throws Exception {
                                                        UIECMAdminPortlet portlet =
 event.getSource().getAncestorOfType(UIECMAdminPortlet.class);
                                            UIECMAdminWorkingArea uiWorkingArea =
 portlet.getChild(UIECMAdminWorkingArea.class);
      uiWorkingArea.setChild(UIWCMCachePanel.class) ;
      event.getRequestContext().addUIComponentToUpdateByAjax(uiWorkingArea);
      }
      }
      }
```

3. Create the *src/main/java* directory and launch *mvn eclipse:eclipse*. You can then launch your eclipse and import this new project.

4. Create a new configuration file called *conf/portal/configuration.xml* to register your action (see *Register your UI Action*) with the *org.exoplatform.webui.ext.UIExtensionManager* service.

Register your UI Action

To register your UI Action, do as follows:

1. Create the code:

```
<configuration xmlns="http://www.exoplaform.org/xml/ns/kernel_1_1.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_1.xsd http://www.exoplaform.org/xml/ns/kernel_1_1.xsd">

  <external-component-plugins>
    <target-component>org.exoplatform.webui.ext.UIExtensionManager</target-component>
    <component-plugin>
      <name>Add.Actions</name>
      <set-method>registerUIExtensionPlugin</set-method>
      <type>org.exoplatform.webui.ext.UIExtensionPlugin</type>
      <init-params>
        <object-param>
          <name>CacheView</name>
          <object type="org.exoplatform.webui.ext.UIExtension">
            <field name="type">
              <string>org.exoplatform.ecm.dms.UIECMAdminControlPanel</string>
            </field>
            <field name="name">
              <string>CacheView</string>
            </field>
            <field name="category">
              <string>GlobalAdministration</string>
            </field>
            <field name="component">
           <string>org.exoplatform.wcm.component.cache.UIWCMCacheComponent</string>
            </field>
          </object>
        </object-param>
        <object-param>
          <name>UIWCMCacheManager</name>
          <object type="org.exoplatform.webui.ext.UIExtension">
            <field name="type">
              <string>org.exoplatform.ecm.dms.UIECMAdminControlPanel</string>
            </field>
            <field name="name">
              <string>UIWCMCacheManager</string>
```

```
                </field>
                <field name="category">
                   <string>GlobalAdministration</string>
                </field>
                <field name="component">
             <string>org.exoplatform.wcm.manager.cache.UIWCMCacheManagerComponent</
string>
                </field>
                <field name="extendedFilters">
                   <collection type="java.util.ArrayList">
                      <value>
                         <object type="org.exoplatform.webui.ext.filter.impl.UserACLFilter">
                            <field name="permissions">
                               <collection item-type="java.lang.String" type="java.util.ArrayList">
                                  <value>
                                     <string>*:/platform/administrators</string>
                                  </value>
                               </collection>
                            </field>
                         </object>
                      </value>
                   </collection>
                </field>
             </object>
          </object-param>
       </init-params>
    </component-plugin>
 </external-component-plugins>
</configuration>
```

Note

With this configuration, only the users with the **:/platform/administrators* membership have the right to access the CacheView item.

2. Launch **mvn clean install**.

3. Copy the resource bundle.

Note

All resources can be located in the *src/main/resource* package because the resources (*.xml, images, conf file) and the code are separated. This is very useful in a hierarchical structure.

Create *ExamplePortlet_en.xml* with the following content and add it to the *src/main/resource* package:

```
<bundle>
                                                                                <!--

 # org.exoplatform.wcm.component.cache.UIWCMCacheForm
                                                                                  #

 -->

   <UIWCMCacheForm>
      <action>
         <Cancel>Cancel</Cancel>
         <Save>Save</Save>
         <Clear>Clear the cache</Clear>
      </action>
      <label>
         <maxsize>Max size :</maxsize>
         <livetime>Live time in sec :</livetime>
               <isCacheEnable>Cache enabled(should always be on production enviroment)</
isCacheEnable>
         <hit>Hit count :</hit>
         <currentSize>Current size</currentSize>
         <miss>Miss count :</miss>
      </label>
   </UIWCMCacheForm>

                                                                                <!--

 # org.exoplatform.wcm.manager.cache.UIWCMCacheManagerForm
                                                                                  #

 -->

   <UIWCMCacheManagerForm>
      <action>
         <Cancel>Cancel</Cancel>
         <Save>Save</Save>
         <Clear>Clear the cache</Clear>
      </action>
      <label>
```

```
        <cacheModify>Cache to modify :</cacheModify>
        <maxsize>Max size :</maxsize>
        <livetime>Live time in sec :</livetime>
              <isCacheEnable>Cache enabled(should always be on production enviroment)</
isCacheEnable>
        <hit>Hit count :</hit>
        <currentSize>Current size</currentSize>
        <miss>Miss count :</miss>
      </label>
   </UIWCMCacheManagerForm>

   <UIECMAdminControlPanel>
      <tab>
        <label>
          <GlobalAdministration>Global Administration</GlobalAdministration>
        </label>
      </tab>
      <label>
        <UIWCMCache>WCM Cache</UIWCMCache>
        <UIWCMCachePanel>WCM Cache Administration</UIWCMCachePanel>
        <UIWCMCacheManager>Managing Caches</UIWCMCacheManager>
       <UIWCMCacheManagerPanel>WCM Cache Management</UIWCMCacheManagerPanel>
      </label>
   </UIECMAdminControlPanel>
</bundle>
```

You must add the following content to *configuration.xml* to register the resource bundle.

Note

By being added this configuration, the resource bundle has been completely separated from the original system. This is clearly useful for you to get an independent plugin.

```
<external-component-plugins>
   <!-- The full qualified name of the ResourceBundleService -->
        <target-component>org.exoplatform.services.resources.ResourceBundleService</target-
component>
   <component-plugin>
      <!-- The name of the plugin -->
      <name>ResourceBundle Plugin</name>
       <!-- The name of the method to call on the ResourceBundleService in order to register the
 ResourceBundles -->
      <set-method>addResourceBundle</set-method>
```

```
      <!-- The full qualified name of the BaseResourceBundlePlugin -->
      <type>org.exoplatform.services.resources.impl.BaseResourceBundlePlugin</type>
      <init-params>
        <values-param>
          <name>init.resources</name>
          <description>Store the following resources into the db for the first launch</description>
          <value>locale.portlet.cache.ExamplePortlet</value>
        </values-param>
        <values-param>
          <name>portal.resource.names</name>
          <description>The properties files of the portal , those file will be merged
            into one ResoruceBundle properties
          </description>
          <value>locale.portlet.cache.ExamplePortlet</value>
        </values-param>
      </init-params>
    </component-plugin>
</external-component-plugins>
```

Run your own UI extension sample

Do as follows:

1. Run Tomcat.

2. Sign in as "root".

3. Go to the ECM Administration. You will see that a new extension has been already added to ECM Administrator.

Authoring Extension

Publication add-ons for eXo Content.

Extended Publication Plugin

States

This extended publication has new states and new profiles that are enabled in eXo Content.

- Profiles
  - Author: This profile can edit a content and mark this content as redacted.
  - Approver: This profile approves a pending content (marked by the Author).

  - Publisher: This profile publishes contents or marks them as "Ready for publication" in multi-server mode.
  - Archiver: An administrative profile which moves contents to an archive storage.
- States
  - enrolled: It is a pure technical state, generally used for content creation.
  - draft (Author): Content is in editing phase.
  - pending (Author): The author validates the content.
  - approved (Approver): A content is approved by the manager.
  - inreview (Manager): This state can be used when a second approval state is needed (for i18 translation for example).
  - staged (Publisher): A content is ready for publication (multi-server mode).
  - published (Publisher or Automatic): A content is published and visible in the **Live** mode.
  - unpublished (Publisher or Automatic): A content is not visible in the **Live** mode.
  - obsolete: A content can still be published but it is not in an editing lifecycle anymore.
  - archived (Automatic): A content is archived and ready to be moved in the archive workspace if enabled.

Start/End publication dates

In most cases, you do not want to publish a content directly, but at a defined date and you can also want the content to be unpublished automatically after that. New properties are added to the new publication plugin, that allows you to manage this:

- publication:startPublishedDate
- publication:endPublishedDate

The eXo Content rendering engine does not know anything about publication dates, so another service needs to manage that. When the publisher sets start/end publication dates, he can "stage" the content. The content will go automatically to the "published" state when the start date arrives and to the "unpublished" state after end date. A cron job checks every hour (or less) all contents which need to be published (the start date in the past and the "staged" state) or unpublished (the end date in the past and the "published" state).

Thus, the publication dates are not mandatory and a content can go to:

- Staged: in multi-server mode, the publisher can only put the content to the "staged" state and wait for auto-publication.

- Published: in single-server mode, the publisher can directly publish a content (with or without publication dates).

New Publication Mixin

```
<nodeType hasOrderableChildNodes="false" isMixin="true"
 name="publication:authoringPublication" primaryItemName="">
  <supertypes>
    <supertype>publication:stateAndVersionBasedPublication</supertype>
  </supertypes>
  <propertyDefinitions>
        <propertyDefinition autoCreated="false" mandatory="true" multiple="false"
      name="publication:startPublishedDate" onParentVersion="IGNORE" protected="false"
 requiredType="Date">
     <valueConstraints/>
    </propertyDefinition>
        <propertyDefinition autoCreated="false" mandatory="true" multiple="false"
      name="publication:endPublishedDate" onParentVersion="IGNORE" protected="false"
 requiredType="Date">
     <valueConstraints/>
    </propertyDefinition>
  </propertyDefinitions>
</nodeType>
```

Publication plugin UI:

Note that some labels containing special or non-ASCII characters could not be well displayed in the publication UI. You can extend the width of the current UI State button by adding:

```
.UIPublicationPanel .StatusTable .ActiveStatus {
  width: 75px !important;
}
```

Also, for the publication date inputs, UIPublicationPanel should not initialize the dates to any default value. The publishing and unpublish CRON jobs will do this:

- A staged document with null publication start date is published instantly.
- A document with null publication end date is published forever.

See the export section for more information about the CRON jobs.

Publication Manager

The Publication Manager manages lifecycles and contexts in the eXo Content platform. It allows to manages different lifecycles based on different publication plugin in the platform.

```
public interface PublicationManager {

 public List<Lifecycle> getLifecycles();

 public List<Context> getContexts();

 public Context getContext(String name);

 public Lifecycle getLifecycle(String name);

 public List<Lifecycle> getLifecyclesFromUser(String remoteUser, String state);
}
```

In which:

- **getLifecycles**: returns a list of lifecycles (see below), with lifecycle name, publication plugin involved and possible states.
- **getContexts**: returns a list of context, with name, related Lifecycle and other properties (see below).
- **getContext**: returns a context by its name.
- **getLifecycle**: returns a lifecycle by its name.
- **getLifecycleFromUser**: returns a list of lifecycles in which the user has rights (based on membership property).

Lifecycle

A lifecycle is defined by a simple vertical workflow with steps (states) and profiles (membership). Each lifecycle is related to a Publication plugin (compliant with the JBPM or Bonita business processes).

For example: *Two lifecycles with/without states*

```
<external-component-plugins>
      <target-component>org.exoplatform.services.wcm.publication.PublicationManager</target-component>
```

```
  <component-plugin>
    <name>AddLifecycle</name>
    <set-method>addLifecycle</set-method>
    <type>org.exoplatform.services.wcm.publication.lifecycles.StatesLifecyclePlugin</type>
    <init-params>
      <object-param>
        <name>lifecyles</name>
      <object type="org.exoplatform.services.wcm.publication.lifecycles.impl.LifecyclesConfig">
          <field name="lifecycles">
            <collection type="java.util.ArrayList">
              <value>
                                                                    <object
type="org.exoplatform.services.wcm.publication.lifecycles.impl.LifecyclesConfig$Lifecycle">
                    <field name="name">
                      <string>lifecycle1</string>
                    </field>
                    <field name="publicationPlugin">
                      <string>States and versions based publication</string>
                    </field>
                  </object>
                </value>
                <value>
                                                                    <object
type="org.exoplatform.services.wcm.publication.lifecycles.impl.LifecyclesConfig$Lifecycle">
                    <field name="name">
                      <string>lifecycle2</string>
                    </field>
                    <field name="publicationPlugin">
                      <string>Authoring publication</string>
                    </field>
                    <field name="states">
                      <collection type="java.util.ArrayList">
                        <value>
                                                                    <object
type="org.exoplatform.services.wcm.publication.lifecycles.impl.LifecyclesConfig$State">
                            <field name="state">
                              <string>draft</string>
                            </field>
                            <field name="memberships">
                              <collection type="java.util.ArrayList">
                                <value>
                                  <string>author:/CA/communicationDG</string>
                                </value>
                                <value>
```

```
                                        <string>author:/CA/alerteSanitaire</string>
                                    </value>
                                    <value>
                                        <string>author:/CA/alerteInformatique</string>
                                    </value>
                                    <value>
                                        <string>author:/CA/informations</string>
                                    </value>
                                </collection>
                            </field>
                        </object>
                    </value>
                    <value>
                                                                                <object
type="org.exoplatform.services.wcm.publication.lifecycles.impl.LifecyclesConfig$State">
                            <field name="state">
                                <string>pending</string>
                            </field>
                            <field name="membership">
                                <string>author:/platform/web-contributors</string>
                            </field>
                        </object>
                    </value>
                    <value>
                                                                                <object
type="org.exoplatform.services.wcm.publication.lifecycles.impl.LifecyclesConfig$State">
                            <field name="state">
                                <string>approved</string>
                            </field>
                            <field name="membership">
                                <string>manager:/platform/web-contributors</string>
                            </field>
                        </object>
                    </value>
                    <value>
                                                                                <object
type="org.exoplatform.services.wcm.publication.lifecycles.impl.LifecyclesConfig$State">
                            <field name="state">
                                <string>staged</string>
                            </field>
                            <field name="membership">
                                <string>publisher:/platform/web-contributors</string>
                            </field>
                        </object>
```

```
                                </value>
                                <value>
                                                                                <object
type="org.exoplatform.services.wcm.publication.lifecycles.impl.LifecyclesConfig$State">
                                    <field name="state">
                                        <string>published</string>
                                    </field>
                                    <field name="membership">
                                        <string>automatic</string>
                                    </field>
                                </object>
                                </value>
                            </collection>
                        </field>
                    </object>
                </value>
                <value>
                                                                                <object
type="org.exoplatform.services.wcm.publication.lifecycles.impl.LifecyclesConfig$Lifecycle">
                        <field name="name">
                            <string>lifecycle3</string>
                        </field>
                        <field name="publicationPlugin">
                            <string>Authoring publication</string>
                        </field>
                        <field name="states">
                            <collection type="java.util.ArrayList">
                                <value>
                                                                                <object
type="org.exoplatform.services.wcm.publication.lifecycles.impl.LifecyclesConfig$State">
                                    <field name="state">
                                        <string>draft</string>
                                    </field>
                                    <field name="membership">
                                        <string>author:/platform/web-contributors</string>
                                    </field>
                                </object>
                                <value>
                                                                                <object
type="org.exoplatform.services.wcm.publication.lifecycles.impl.LifecyclesConfig$State">
                                    <field name="state">
                                        <string>published</string>
                                    </field>
```

```
                                    <field name="memberships">
                                      <collection type="java.util.ArrayList">
                                        <value>
                                          <string>publisher:/CA/communicationDG</string>
                                        </value>
                                        <value>
                                          <string>publisher:/CA/alerteSanitaire</string>
                                        </value>
                                        <value>
                                          <string>publisher:/CA/alerteInformatique</string>
                                        </value>
                                        <value>
                                          <string>publisher:/CA/informations</string>
                                        </value>
                                      </collection>
                                    </field>
                                  </object>
                                </value>
                              </collection>
                            </field>
                          </object>
                        </value>
                      </collection>
                    </field>
                  </object>
                </object-param>
              </init-params>
            </component-plugin>
</external-component-plugins>
```

In the last example, there are three lifecycles:

- Lifecycle 1: Based on *StatesAndVersionsPublicationPlugin* .
  - This allows to be backward compliant with older eXo Content releases. If all your site contents are using an existing plugin, you can create a lifecycle for it and it will work.
  - For new instances, you should use the new plugin with dynamic states capabilities.
- Lifecycle 2: Based on *AuthoringPublicationPlugin* .
  - Visibility: Define only the "visible" steps. In this example, there is no step for 'enrolled'. Even if this step exists, it will not be displayed in the UI.
  - Automatic: Set a step as "automatic". In this mode, the step will be visible in the UI but it will be managed by the system (e.g. a cron job).

- Lifecycle 3: Simulates the *StatesAndVersionsPublicationPlugin* plugin. Note that this simple lifecycle will work in a single server configuration.

Listen to a lifecycle

When a state is changed, you can broadcast an event to add features. The event could look like this:

```
listenerService.broadcast(AuthoringPlugin.POST_UPDATE_STATE_EVENT, null, node);
```

Listener declaration could look like this:

```
<external-component-plugins>
  <target-component>org.exoplatform.services.listener.ListenerService</target-component>
  <component-plugin>
    <name>PublicationService.event.postUpdateState</name>
    <set-method>addListener</set-method>


type>
      <description>this listener will be called every time a content changes its current state</
description>
  </component-plugin>
</external-component-plugins>
```

Context

A context is defined by simple rules. In eXo Content, you can select to enroll the content in a specific lifecycle (for example, publication plugin) based on context parameters. There are three parameters used to define contexts:

- Remote User: The current user who can create/edit the content.
- Current site name: The site from where the content is created (not the storage but the navigation).
- Node: The node which you want to enroll.

From these parameters, you can easily connect and define contexts based on:

- Membership: Does the current user have this membership?
- Site: On this particular site, you want to enroll contents in a specific lifecycle.

- Path: You can enroll contents in the lifecycles based on their path (from the Node).
- Type of content: You can enroll contents in the lifecycles based on their nodetype (from the Node).

Because each site has a content storage (categories + physical storage), you can select the right lifecycle for the right storage/site. To avoid conflicts on contexts, you can set a priority (the less is the best).

For example, **Different Contexts**:

```
<external-component-plugins>
      <target-component>org.exoplatform.services.wcm.publication.PublicationManager</target-component>
   <component-plugin>
      <name>AddContext</name>
      <set-method>addContext</set-method>
      <type>org.exoplatform.services.wcm.publication.context.ContextPlugin</type>
      <init-params>
         <object-param>
            <name>contexts</name>
            <object type="org.exoplatform.services.wcm.publication.context.impl.ContextConfig">
               <field name="contexts">
                  <collection type="java.util.ArrayList">
                     <value>
                                                                                    <object type="org.exoplatform.services.wcm.publication.context.impl.ContextConfig$Context">
                           <field name="name">
                              <string>contextdefault</string>
                           </field>
                           <field name="priority">
                              <string>200</string>
                           </field>
                           <field name="lifecycle">
                              <string>lifecycle1</string>
                           </field>
                        </object>
                                                                                    <object type="org.exoplatform.services.wcm.publication.context.impl.ContextConfig$Context">
                           <field name="name">
                              <string>context1</string>
                           </field>
                           <field name="priority">
                              <string>100</string>
```

```
                        </field>
                        <field name="lifecycle">
                            <string>lifecycle1</string>
                        </field>
                        <field name="membership">
                            <string>*:/platform/web-contributors</string>
                        </field>
                        <field name="site">
                            <string>acme</string>
                        </field>
                        <field name="path">
                      <string>repository:collaboration:/sites content/live/acme/categories</string>
                        </field>
                    </object>
                                                                                <object
type="org.exoplatform.services.wcm.publication.context.impl.ContextConfig$Context">
                        <field name="name">
                            <string>context2</string>
                        </field>
                        <field name="priority">
                            <string>100</string>
                        </field>
                        <field name="lifecycle">
                            <string>lifecycle1</string>
                        </field>
                        <field name="site">
                            <string>classic</string>
                        </field>
                    </object>
                                                                                <object
type="org.exoplatform.services.wcm.publication.context.impl.ContextConfig$Context">
                        <field name="name">
                            <string>context3</string>
                        </field>
                        <field name="priority">
                            <string>80</string>
                        </field>
                        <field name="lifecycle">
                            <string>lifecycle3</string>
                        </field>
                        <field name="membership">
                            <string>manager:/company/finances</string>
                        </field>
                        <field name="path">
```

```
                        <string>repository:collaboration:/documents/company/finances</string>
                      </field>
                    </object>
                                                                                    <object
type="org.exoplatform.services.wcm.publication.context.impl.ContextConfig$Context">
                      <field name="name">
                        <string>context4</string>
                      </field>
                      <field name="priority">
                        <string>50</string>
                      </field>
                      <field name="lifecycle">
                        <string>lifecycle4</string>
                      </field>
                      <field name="memberships">
                        <collection type="java.util.ArrayList">
                          <value>
                            <string>manager:/CA/communicationDG</string>
                          </value>
                          <value>
                            <string>manager:/CA/alerteSanitaire</string>
                          </value>
                          <value>
                            <string>manager:/CA/alerteInformatique</string>
                          </value>
                          <value>
                            <string>manager:/CA/informations</string>
                          </value>
                        </collection>
                      </field>
                      <field name="path">
                        <string>repository:collaboration:/documents/company/finances</string>
                      </field>
                      <field name="nodetype">
                        <string>exo:article</string>
                      </field>
                    </object>
                  </value>
                </collection>
              </field>
            </object>
          </object-param>
        </init-params>
      </component-plugin>
```

```
</external-component-plugins>
```

The logic is very simple. When creating a content, it should be attached a lifecycle with the lifecyle priority:

- *context4* is the most important (priority=50): you will enroll the content in the lifecycle 'lifecycle4' if:
  - The content creator has the '`manager:/company/finances`' membership.
  - The content is stored in '*repository:collaboration:/documents/company/finances*' or any subfolders.
  - The content is a '`exo:article`'.
- If not, you will continue with *context3*.

The logic is very simple. When you create a content, go lifecycle by lifecycle starting with the better priority:

- *context4* is the most important (priority=50): you will enroll the content in the lifecycle 'lifecycle4' if:
  - The content creator has the '`manager:/company/finances`' membership.
  - The content is stored in '*repository:collaboration:/documents/company/finances*' or any subfolders.
  - The content is a '`exo:article`'.
- If not, you will continue with *context3*.

Note

The contexts will be used only when the content is created and when you want to enroll it in a lifecycle for the first time. Once you have the corresponding lifecycle, you will set the lifecycle inside the content (see *New Authoring Mixin*) and the context service will not be called again for this content.

New Authoring Mixin

```
<nodeType hasOrderableChildNodes="false" isMixin="true" name="publication:authoring"
 primaryItemName="">
  <propertyDefinitions>
```

```
                        <propertyDefinition autoCreated="false" mandatory="false"
  multiple="false" name="publication:lastUser" onParentVersion="IGNORE" protected="false"
 requiredType="String">
    <valueConstraints/>
   </propertyDefinition>
                        <propertyDefinition autoCreated="false" mandatory="false"
  multiple="false" name="publication:lifecycle" onParentVersion="IGNORE" protected="false"
 requiredType="String">
    <valueConstraints/>
   </propertyDefinition>
 </propertyDefinitions>
</nodeType>
```

When adding the content in a lifecycle, set the `publication:lifecycle` property with the corresponding lifecycle.

Note

A content can be in one lifecycle only.

Each time you change from one state to another, set the user who changed the state in *publication:lastUser*.

Querying based on publication status:

By adding this mixin to contents, you can access contents by simple queries based on the current user profile. For example:

- All your draft contents:
  - query: select * from `nt:base` where `publication:currentState`='draft' and `publication:lastUser`='benjamin'.
- All the contents you have to approve.
  - call: `PublicationManager.getLifecycles('benjamin','approved')` => returns lifecycles where you can go to the 'approved' state.
  - query: select * from `nt:base` where publication:currentState='pending' and (publication:lifecycle='lifecycle1' or publication:lifecycle='lifecycle3').
- All the content that will be published tomorrow.
  - query: select * from `nt:base` where `publication:currentState`='staged' and `publication:startPublishedDate`='xxxx'.

Public REST APIs

ThumbnailRESTService

Service name	Service URL	Location	Description
ThumbnailRESTService	{portalname}/{restcontextname}/thumbnailImage/	**Maven groupId**: org.exoplatform.ecms **ArtifactId**: exo-ecms-core-services	Return a responding data as a thumbnail image.

Note

- {portalname}: The name of the portal.
- {restcontextname}: The context name of rest webapplication which is deployed to the "{portalname}" portal.

- **APIs usage**:

Name	Service URL endpoint	Parameters	Values	Description
getThumbnailImage	{portalname}/{restcontextname}/thumbnailImage/medium/{repoName}/{workspaceName}/{nodePath}/	repoName workspaceName nodePath	String String String	Return an image with the medium size (64x64).
getLargeImage	{portalname}/{restcontextname}/thumbnailImage/large/{repoName}/{workspaceName}/{nodePath}/	repoName workspaceName nodePath	String String String	Return an image with the large size (300x300).
getSmallImage	{portalname}/{restcontextname}/thumbnailImage/small/{repoName}/{workspaceName}/{nodePath}/	repoName workspaceName nodePath	String String String	Return an image with the small size (32x32).

Name	Service URL endpoint	Parameters	Values	Description
getCustomImage	{portalname}/ {restcontextname}/ thumbnailImage/ custom/{size}/ {repoName}/ {workspaceName}/ {nodePath}/	size repoName workspaceName nodePath	String String String String	Return an image with the custom size.
getOriginImage	{portalname}/ {restcontextname}/ thumbnailImage/ origin/ {repoName}/ {workspaceName}/ {nodePath}/	repoName workspaceName nodePath	String String String	Return an image with the original size.

RssConnector

Service name	Service URL	Location	Description
RssConnector	{portalname}/ {restcontextname}/ feed/	**Maven groupId**: org.exoplatform.ecms **ArtifactId**: exo-ecms-core-connector	Generate an RSS feed.

- **APIs usage**:

Name	Service URL endpoint	Parameters	Values	Description
generate	{portalname}/ {restcontextname}/ feed/rss/	repository workspace server siteName title	String String String String String	Generate an RSS feed.

Name	Service URL endpoint	Parameters	Values	Description
		desc	String	
		folderPath	String	
		orderBy	String	
		orderType	String	
		lang	String	
		detailPage	String	
		detailParam	String	
		recursive	String	

FCKCoreRESTConnector

Service name	Service URL	Location	Description
FCKCoreRESTConnector	{portalname}/{restcontextname}//fckconnector/jcr/	**Maven groupId**: org.exoplatform.ecms **ArtifactId**: exo-ecms-core-connector	Get a list of files and folders, and create a folder and upload files.

- **APIs usage**:

Name	Service URL endpoint	Parameters	Values	Description
getFoldersAndFiles	{portalname}/{restcontextname}/fckconnector/jcr/getFoldersAndFiles/{repositoryName}/{workspaceName}/{currentFolder}/{command}/{type}/	repositoryName	String	Return the folders and the files in the current folder.
		workspaceName	String	
		currentFolder	String	
		command	String	

Name	Service URL endpoint	Parameters	Values	Description
		type	String	
createFolder	{portalname}/ {restcontextname}/ fckconnector/jcr/ createFolder/ {repositoryName}/ {workspaceName}/ {currentFolder}/ {newFolderName}/ {language}/	repositoryName workspaceName currentFolder newFolderName language	String String String String String	Create a folder under the current folder.
uploadFile	{portalname}/ {restcontextname}/ fckconnector/jcr/ uploadFile/ upload/	servletRequest	HttpServletRequest	Uploads a file with the HttpServletRequest.
processUpload	{portalname}/ {restcontextname}/ fckconnector/jcr/ uploadFile/ control/ {repositoryName}/ {workspaceName}/ {currentFolder}/ {action}/ {language}/ {fileName}/ {uploadId}/	repositoryName workspaceName currentFolder action language fileName uploadId	String String String String String String String	Control the process of uploading a file, such as aborting, deleting or progressing the file.

ResourceBundleConnector

Service name	Service URL	Location	Description
ResourceBundleConnector	{portalname}/ {restcontextname}/ bundle/	**Maven groupId**: org.exoplatform.ecms **ArtifactId**: exo-ecms-core-connector	Get the bundle basing on the key and the locale.

- **APIs usage**:

Name	Service URL endpoint	Parameters	Values	Description
getBundle	{portalname}/ {restcontextname}/ bundle/ getBundle/{key}/ {locale}/	key locale	String String	Get the bundle basing on the key and the locale.

VoteConnector

Service name	Service URL	Location	Description
VoteConnector	{portalname}/ {restcontextname}/ contents/vote/	**Maven groupId**: org.exoplatform.ecms **ArtifactId**: exo-ecms-core-connector	Return and set a vote value of a given node in the sent parameter.

- **APIs usage**:

Name	Service URL endpoint	Parameters	Values	Description
postVote	{portalname}/ {restcontextname}/ contents/vote/ postVote/ {repositoryName}/ {workspaceName}/ {jcrPath}/{vote}/ {lang}/	repositoryName workspaceName jcrPath vote lang	String String String String String String	Set a vote value for a given content.
getVote	{portalname}/ {restcontextname}/ contents/vote/ getVote/ {repositoryName}/ {workspaceName}/ {jcrPath}/	repositoryName workspaceName jcrPath	String String String	Return a vote value for a given content.

DriverConnector

Service name	Service URL	Location	Description
DriverConnector	{portalname}/ {restcontextname}/ wcmDriver/	**Maven groupId**: org.exoplatform.ecms **ArtifactId**: exo-ecms-core-connector	Return a drive list, a folder list and a document list in a specified location for a given user. Also, it processes the file uploading action.

- **APIs usage**:

Name	Service URL endpoint	Parameters	Values	Description
getDrivers	{portalname}/ {restcontextname}/ wcmDriver/ getDrivers/	lang	String	Return a list of drives for the current user.
getFoldersAndFiles	{portalname}/ {restcontextname}/ wcmDriver/ getFoldersAndFiles/	driverName currentFolder currentPortal repositoryName workspaceName filterBy	String String String String String String	Return all folders and files in a given location.
uploadFile	{portalname}/ {restcontextname}/ wcmDriver/ uploadFile/ upload/	uploadId	String	Uploads a file.
processUpload	{portalname}/ {restcontextname}/ wcmDriver/ uploadFile/ control/	repositoryName workspaceName	String String	Control the process of uploading a file, such as aborting, deleting

Name	Service URL endpoint	Parameters	Values	Description
		driverName	String	or processing the file.
		currentFolder	String	
		currentPortal	String	
		userId	String	
		jcrPath	String	
		action	String	
		language	String	
		fileName	String	
		uploadId	String	

GadgetConnector

Service name	Service URL	Location	Description
GadgetConnector	{portalname}/{restcontextname}/wcmGadget/	**Maven groupId**: org.exoplatform.ecms **ArtifactId**: exo-ecms-core-connector	Instantiate a new gadget connector.

- **APIs usage**:

Name	Service URL endpoint	Parameters	Values	Description
getFoldersAndFiles	{portalname}/{restcontextname}/wcmGadget/getFoldersAndFiles/{currentFolder}/{lang}/{host}/	currentFolder	String	Get the folders and files.
		lang	String	
		host	String	

PortalLinkConnector

Service name	Service URL	Location	Description
PortalLinkConnector	{portalname}/ {restcontextname}/ portalLinks/	**Maven groupId**: org.exoplatform.ecms **ArtifactId**: exo-ecms-core-connector	Return a page URI for a given location.

- **APIs usage**:

Name	Service URL endpoint	Parameters	Values	Description
getPageURI	{portalname}/ {restcontextname}/ portalLinks/ getFoldersAndFiles/	currentFolder	String String String	Get the page URI.

GetEditedDocumentRESTService

Service name	Service URL	Location	Description
GetEditedDocumentRESTService	{portalname}/ {restcontextname}/ presentation/ document/edit/	**Maven groupId**: org.exoplatform.ecms **ArtifactId**: exo-ecms-core-publication	Return the latest edited documents.

- **APIs usage**:

Name	Service URL endpoint	Parameters	Values	Description
getLastEditedDoc	{portalname}/ {restcontextname}/ presentation/ document/edit/ {repository}/	repository	String	Return the latest edited documents.

PublicationGetDocumentRESTService

Service name	Service URL	Location	Description
PublicationGetDocumentRESTService	{portalname}/ {restcontextname}/	**Maven groupId**: org.exoplatform.ecms	Return a list of published documents.

Service name	Service URL	Location	Description
	publication/ presentation/	**ArtifactId**: exo-ecms-core-publication	

- **APIs usage**:

Name	Service URL endpoint	Parameters	Values	Description
getPublishDocument	{portalname}/ {restcontextname}/ publication/ presentation/ {repository}/ {workspace}/ {state}/	repository workspace state showItems	String String String String	Return a list of published document by the default plugin.
getPublishedListDocument	{portalname}/ {restcontextname}/ publication/ presentation/ {repository}/ {workspace}/ {publicationPluginName}/ {state}/	repository workspace publicationPluginName state showItems	String String String String String String	Return a list of published documents by a specific plugin.

FavoriteRESTService

Service name	Service URL	Location	Description
FavoriteRESTService	{portalname}/ {restcontextname}/ favorite/	**Maven groupId**: org.exoplatform.ecms **ArtifactId**: exo-ecms-core-services	Return a list of favourite documents of a given user.

- **APIs usage**:

Name	Service URL endpoint	Parameters	Values	Description
getFavoriteByUser	{portalname}/ {restcontextname}/ favorite/all/ {repoName}/ {workspaceName}/ {userName}	repoName workspaceName userName showItems	String String String String	Return a list of favourite documents of a given user.

RESTImagesRendererService

Service name	Service URL	Location	Description
RESTImagesRendererService	{portalname}/ {restcontextname}/ images/	**Maven groupId**: org.exoplatform.ecms **ArtifactId**: exo-ecms-core-services	Get the image binary data of a given image node.

- **APIs usage**:

Name	Service URL endpoint	Parameters	Values	Description
serveImage	{portalname}/ {restcontextname}/ images/ {repositoryName}/ {workspaceName}/ {nodeIdentifier}/	repositoryName workspaceName nodeIdentifier param	String String String String	Get the image binary data of a given image node.

LifecycleConnector

Service name	Service URL	Location	Description
LifecycleConnector	{portalname}/ {restcontextname}/ authoring/	**Maven groupId**: org.exoplatform.ecms **ArtifactId**: exo-ecms-ext-authoring-services	Return a list of contents in a given state range of the publication lifecycle.

- **APIs usage**:

Name	Service URL endpoint	Parameters	Values	Description
byState	{portalname}/ {restcontextname}/ authoring/ bystate/	fromstate user lang workspace json	String String String String String	Return a list of contents from the given state to the last state.
toState	{portalname}/ {restcontextname}/ authoring/ toState/	fromstate tostate user lang workspace json	String String String String String String	Return a list of contents from the beginning state to the last state.
byDate	{portalname}/ {restcontextname}/ authoring/ byDate/	fromstate date lang workspace json	String String String String String String	Return a list of contents from the given beginning state and published before the given date.

CopyContentFile

Service name	Service URL	Location	Description
CopyContentFile		**Maven groupId**: org.exoplatform.ecms	Copy a file.

Service name	Service URL	Location	Description
	{portalname}/ {restcontextname}/ copyfile/	**ArtifactId**: exo-ecms-ext-authoring-services	

- **APIs usage**:

Name	Service URL endpoint	Parameters	Values	Description
copyFile	{portalname}/ {restcontextname}/ copyfile/copy/	param - the file information	String	Copy a file.

PDFViewerRESTService

Service name	Service URL	Location	Description
PDFViewerRESTService	{portalname}/ {restcontextname}/ pdfviewer// {repoName}/ {workspaceName}/ {pageNumber}/ {rotation}/{scale}/ {uuid}/	**Maven groupId**: org.exoplatform.ecms **ArtifactId**: ecms-core-viewer	Return the pdf content to display on the web page

- **APIs usage**:

Name	Service URL endpoint	Parameters	Values	Description
getCoverImage	{portalname}/ {restcontextname}/ pdfviewer// {repoName}/ {workspaceName}/ {pageNumber}/ {rotation}/{scale}/ {uuid}/	repoName workspaceName uuid pageNumber rotation scale	String String String String String	Return a thumbnail image for a pdf document.

Public Java APIs

TaxonomyService

Taxonomy service is used to work with taxonomies. In this service, there are many functions which enable you to add, find, or delete taxonomies from a node.

Method	Param	Return	Description
getTaxonomyTree(String repository, String taxonomyName, boolean system) throws ReposioryException;	`repository`: The name of repository. `taxonomyName`: The name of the taxonomy. `system`: Indicates whether the nodes must be retrieved using a session system or user session.	Node	Return the root node of the given taxonomy tree.
getTaxonomyTree(String repository, String taxonomyName) throws RepositoryException;	`repository`: The name of repository. `taxonomyName`: The name of the taxonomy.	Node	Return the root node of the given taxonomy tree with the user session.
getAllTaxonomyTrees(String repository, boolean system) throws RepositoryException;	`repository`: The name of repository. `system`: Indicates whether the nodes must be retrieved using a session system or user session.	List<Node>	Return the list of all the root nodes of the taxonomy tree available.
getAllTaxonomyTrees(String repository) throws RepositoryException;	`repository`: The name of repository.	List<Node>	Return the list of all the root nodes of the taxonomy tree available with the user session.

Method	Param	Return	Description
hasTaxonomyTree(String repository, String taxonomyName) throws RepositoryException;	`repository`: The name of repository. `taxonomyName`: The name of the taxonomy.	boolean	Check if a taxonomy tree with the given name has already been defined.
addTaxonomyTree(Node taxonomyTree) throws RepositoryException, TaxonomyAlreadyExistsException;	`taxonomyTree`: The taxonomy tree to define.	void	Define a node as a new taxonomy tree.
updateTaxonomyTree(String taxonomyName, Node taxonomyTree) throws RepositoryException;	`taxonomyName`: The name of the taxonomy to update. `taxonomyTree`: The taxonomy tree to define.	void	Re-define a node as a taxonomy tree.
removeTaxonomyTree(String taxonomyName) throws RepositoryException	`taxonomyName`: The name of the taxonomy to remove.	void	Remove the taxonomy tree definition.
addTaxonomyNode(String repository, String workspace, String parentPath, String taxoNodeName, String creator) throws RepositoryException, TaxonomyNodeAlreadyExistsException;	`repository`: The name of the repository. `workspace`: The name of the workspace `parentPath`: The place where the taxonomy node will be added. `taxoNodeName`: The name of taxonomy node. `creator`: The name of the user creating this node.	void	Add a new taxonomy node at the given location.

Method	Param	Return	Description
removeTaxonomyNode(String repository, String workspace, String absPath) throws RepositoryException;	`repository`: The name of the repository `workspace`: The name of the workspace. `absPath`: The absolute path of the taxonomy node to remove.	void	Remove the taxonomy node located at the given absolute path.
moveTaxonomyNode(String repository, String workspace, String srcPath, String destPath, String type) throws RepositoryException;	`repository`: The name of the repository. `workspace`: The name of the workspace. `srcPath`: The source path of this taxonomy. `destPath`: The destination path of the taxonomy. `type`: If **type** is equal to **cut**, the process will be cut. If **type** is equal to **copy**, the process will be copied.	void	Copy or cut the taxonomy node from the source path to the destination path. The parameter type indicates if the node must be cut or copied.
hasCategories(Node node, String taxonomyName) throws RepositoryException;	`node`: The node to check. `taxonomyName`: The name of the taxonomy.	boolean	Return true if the given node has categories in the given taxonomy.
hasCategories((Node node, String taxonomyName, boolean system)	`node`: The node to check.	boolean	Return true if the given node has categories in the given taxonomy.

Method	Param	Return	Description
throws RepositoryException;	`taxonomyName`: The name of the taxonomy. `system`: Check system provider or not.		
getCategories (Node node, String taxonomyName) throws RepositoryException;	`node`: The node for which we seek the categories. `taxonomyName`: The name of the taxonomy.	List<Node>	Return all the paths of the categories (relative to the root node of the given taxonomy) which have been associated to the given node for the given taxonomy.
getCategories(Node node, String taxonomyName, boolean system) throws RepositoryException;	`node`: The node for which we seek the categories. `taxonomyName`: The name of the taxonomy. `system`: Check system provider or not.	List<Node>	Return all the paths of the categories(relative to the root node of the given taxonomy) which have been associated to the given node for the given taxonomy.
getAllCategories(Node node) throws RepositoryException;	`node`: The node for which we seek the categories	List<Node>	Return all the paths of the categories which have been associated to the given node.
getAllCategories(Node node, boolean system) throws RepositoryException;	`node`: The node for which we seek the categories `system`: Check system provider or not.	List<Node>	Return all the paths of the categories which have been associated to the given node.
removeCategory(Node node, String taxonomyName,	`node`: The node from which the category is removed.	void	Remove a category to the given node.

Method	Param	Return	Description
String categoryPath) throws RepositoryException;	`taxonomyName`: The name of the taxonomy. `categoryPath`: The path of the category relative to the root node of the given taxonomy		
removeCategory(Node node, String taxonomyName, String categoryPath, boolean system) throws RepositoryException;	`node`: The node from which the category is removed. `taxonomyName`: The name of the taxonomy. `categoryPath`: The path of the category relative to the root node of the given taxonomy. `system`: Check system provider or not.	void	Remove a category to the given node.
addCategories(Node node, String taxonomyName, String[] categoryPaths) throws RepositoryException;	`node`: The node to which we add the categories. `taxonomyName`: The name of the taxonomy. `categoryPaths`: An array of category paths relative to the given taxonomy.	void	Add several categories to the given node.

Method	Param	Return	Description
addCategories(Node node, String taxonomyName, String[] categoryPaths, boolean system) throws RepositoryException;	`node`: The node to which we add the categories. `taxonomyName`: The name of the taxonomy. `categoryPaths`: An array of category paths relative to the given taxonomy. `system`: Check system provider or not.	void	Add several categories to the given node.
addCategory(Node node, String taxonomyName, String categoryPath) throws RepositoryException;	`node`: The node to which we add the category. `taxonomyName`: The name of the taxonomy. `categoryPath`: The path of the category relative to the given taxonomy.	void	Add a new category path to the given node.
addCategory(Node node, String taxonomyName, String categoryPath, boolean system) throws RepositoryException;	`node`: The node to which we add the category. `taxonomyName`: The name of the taxonomy. `categoryPath`: The path of the category relative to the given taxonomy.	void	Add a new category path to the given node.

Method	Param	Return	Description
	`system`: Check system provider or not.		
getTaxonomyTreeDefaultUserPermission();		Map<String, String[]>	Get the default permission for the user in taxonomy tree.
addTaxonomyPlugin(ComponentPlugin plugin);	`plugin`: The plugin to add.	void	Add a new taxonomy plugin to the service.
init(String repository) throws Exception;	`repository`: The name of repository.	void	Initialize all taxonomy plugins that have been already configured in .xml files.
getCategoryNameLengthMax();		String	Get the limited length of the category name.

LinkManager

Supply API to work with the linked node or the link included in a node.

Package org.exoplatform.services.cms.link.LinkManager

Method	Param	Return	Description
createLink(Node parent, String linkType, Node target)throws RepositoryException;	`parent`: The parent node of the link. `linkType`: The primary node type of the link must be a sub-type of exo:symlink, the default value is "exo:symlink" `target`: The target of the link.	Node	Create a new link that is added to the parent node and return the link.
createLink(Node parent, Node target)throws RepositoryException;	`parent`: The parent node of the link to create. `target`: The target of the link.	Node	Create a new node of type exo:symlink, then add it to the parent node and return the link node.

Method	Param	Return	Description
createLink(Node parent, String linkType, Node target, String linkName) throws RepositoryException;	`parent`: The parent node of the link. `inkType`: The primary node type of the link must be a sub-type of exo:symlink, the default value is exo:symlink. `target`: The target of the link. `linkName`: The name of the link.	Node	Create a new link that is added to the parent node and return the link.
updateLink(Node link, Node target)throws RepositoryException;	`link`: The link node to update. `target`: The new target of the link.	Node	Update the target node of the given link.
getTarget(Node link, boolean system) throws ItemNotFoundException, RepositoryException;	`link`: The node of type exo:symlink. `system`: Indicate whether the target node must be retrieved using a session system or user session in case we cannot use the same session as the node link because the target and the link are not in the same workspace.	Node	Get the target node of the given link.
getTarget(Node link)throws ItemNotFoundException, RepositoryException;	`link`: The node of type exo:symlink.	Node	Get the target node of the given link using the user.

Method	Param	Return	Description
isTargetReachable(Node link) throws RepositoryException;	`link`: The node of type exo:symlink.	boolean	Check if the target node of the given link can be reached using the user session.
isTargetReachable(Node link, boolean system)throws RepositoryException;	`link`: The node of type exo:symlink. `system`	boolean	Check if the target node of the given link can be reached using the user session.
isLink(Item item) throws RepositoryException;	`item`: The item to test.	boolean	Indicate whether the given item is a link. @return <code>true</code>: if the node is a link, <code>false</code> otherwise.
getTargetPrimaryNodeType(Node link)throws RepositoryException;	`link`: The node of type exo:symlink.	String	Return the primary node type of the target.
getAllLinks(Node targetNode, String linkType, String repoName)throws Exception	`targetNode`: The target node to get links. `linkType`: The type of link to get. `repoName`: The name of the repository.	List<Node> - the list of link of the target node with given type.	Return all links of the given node.

PublicationManager

PublicationService is to manage the publication.

Method	Param	Return	Description
addLifecycle(ComponentPlugin plugin)	N/A	void	Add plugins context for the publication service.
getLifecycle(String name)	`name`: The name of the wanted lifecycle.	Lifecycle	Get a specific lifecycle with the given name.
getLifecycles()	N/A	List<Lifecycle>	Get all the lifecycles which were added to service instances.

Method	Param	Return	Description
getContext(String name)	`name`	Context	Get a specific context with the given names.
getContexts()	N/A	List<Context>	Get all the contexts which were added to service instances.
getContents(String fromstate, String tostate, String date, String user, String lang , String workspace) throws Exception;	`fromstate/tostate`: The current range state of node. `user`: The last user of node. `lang`: The node's language. `workspace`: The Workspace of the node's location.	List<Node>	Get all the nodes.
getLifecyclesFromUser(String remoteUser, String state);	`remoteUser`: The current user of publication service. `state`: The current state of the node.	List<Lifecycle>	Get all the Lifecycle of a specific user.

WCMComposer

This class is used to get contents inside the WCM product. You should not access contents directly from the JCR on the front side.

In general, this service stands between publication and cache.

Package org.exoplatform.services.wcm.publication.WCMComposer

Method	Param	Return	Description

getContent(String repository, String workspace, String nodeIdentifier, HashMap<String, String> filters, SessionProvider	`repository`
	
	`workspace`
	
	`nodeIdentifier`
		Node	Return contents at the specified path based on filters.

Method	Param	Return	Description
sessionProvider)throws Exception;	`filters` `sessionProvider`: The session provider.		

getContents(String repository, String workspace, String path, HashMap<String, String> filters, SessionProvider sessionProvider)throws Exception;	`repository`
	
	`workspace`
	
	`path`
	
	`filters`
	
	`sessionProvider`: The session provider.
		List<Node>	Return contents at the specified path based on filters.

updateContent(String repository, String workspace, String nodeIdentifier, HashMap<String, String> filters)throws Exception;	`epository`
	
	`workspace`
	
	`path`
	
	`filters`
		boolean	Update content.

getAllowedStatesString mode)throws Exception;	`mode`	List<String>	Return allowed states for a specified mode.
cleanTemplates()throws Exception;	N/A	void	Initialize the template hashmap.
isCached()throws Exception;	N/A	boolean	Check isCache or not.
updateTemplatesSQLFilter()throws Exception;	N/A	String	Update all document nodetypes and write a query cause. It returns a part of the query that allows to search all document nodes and taxonomy links. Return null if there is any exception.

NewFolksonomy

Package org.exoplatform.services.cms.folksonomy.NewFolksonomyService;

Method	Return	Prototype	Description
addPrivateTag(String[] tagsName, Node documentNode, String repository, String workspace, String userName) throws Exception ;	`tagNames`: The array of tag name as the children of tree. `documentNode`: Tag this node by creating a folksonomy link to the node in the tag. `repository`: The repository name. `workspace`: The workspace name. `userName`: The username.	void	Add a private tag to a document. A folksonomy link will be created in a tag node.
addGroupsTag(String[] tagsName, Node documentNode,String repository, String workspace, String[] roles) throws Exception ;	`tagNames`: The array of tag name as the children of tree. `documentNode`: Tag this node by creating a folksonomy link to the node in tag. `repository`: The repository name. `workspace`: The workspace name. `roles`: The user roles.	void	Add a group tag to a document. A folksonomy link will be created in a tag node.
addPublicTag(String treePath, String[] tagsName, Node	`treePath`: The path of folksonomy tree.	void	Add a public tag to a document. A

Method	Return	Prototype	Description
documentNode, String repository, String workspace) throws Exception ;	`tagNames`: The array of the tag name as the children of tree. `documentNode`: Tag this node by creating a folksonomy link to the node in the tag. `repository`: The repository name. `workspace`: The workspace name.		folksonomy link will be created in a tag node.
addSiteTag(String siteName, String[] tagsName, Node node, String repository, String workspace) throws Exception ;	`siteName`: The portal name. `treePath`: The path of folksonomy tree. `tagNames`: The array of the tag name as the children of tree. `documentNode`: Tag this node by creating a folksonomy link to the node in tag. `repository`: The repository name. `workspace`: The workspace name.	void	Add a site tag to a document. A folksonomy link will be created in a tag node.
getAllPrivateTags(String userName, String repository, String	`userName`: The user name.	List<Node>	Get all private tags.

Method	Return	Prototype	Description
workspace) throws Exception ;	`repository`: The repository name. `workspace`: The workspace name.		
getAllPublicTags(String treePath, String repository, String workspace) throws Exception ;	`treePath`: The folksonomy tree path. `repository`: The repository name. `workspace`: The workspace name.	List<Node>	Get all public tags.
getAllGroupTags(String[] roles, String repository, String workspace) throws Exception ;	`roles`: The roles of user. `repository`: The repository name. `workspace`: The workspace name.	List<Node>	Get all tags by groups.
getAllGroupTags(String role, String repository, String workspace) throws Exception ;	`role`: The role of user. `repository`: The repository name. `workspace`: The workspace name.	List<Node>	Get all tags by a group.
getAllSiteTags(String siteName, String repository, String workspace) throws Exception ;	`siteName`: The portal name. `treePath`: Folksonomy tree path. `repository`: The repository name.	List<Node>	Get all tags of Site.

Method	Return	Prototype	Description
	`workspace`: The workspace name.		
getAllDocumentsByTag(String tagPath, String repository, String workspace, SessionProvider sessionProvider) throws Exception ;	`treeName`: The name of folksonomy tree. `tagName`: The name of a tag. `repository`: The repository name.	List<Node>	Get all documents which are stored in a tag and return a list of documents in a tag.
getTagStyle(String tagPath, String repository, String workspace) throws Exception ;	`tagPath` `workspace`: The workspace name. `repository`: The repository name.	String	Get HTML_STYLE_PROP property in styleName node in the repository.
addTagStyle(String styleName, String tagRange, String htmlStyle, String repository, String workspace) throws Exception ;	`styleName`: The style name. `tagRate`: The range of tag numbers. `htmlStyle`: The tag style. `repository`: The repository name. `workspace`: The workspace name.	void	Update the properties TAG_RATE_PROP and HTML_STYLE_PROP, following the values tagRate, htmlStyle for a node in tagPath in repository.
updateTagStyle(String styleName, String tagRange, String htmlStyle, String repository, String workspace) throws Exception ;	`styleName`: The style name. `tagRate`: The range of tag numbers.	void	Update the properties TAG_RATE_PROP and HTML_STYLE_PROP, following the value tagRate, htmlStyle for

Method	Return	Prototype	Description
	`htmlStyle`: The tag style. `repository`: The repository name. `workspace`: The workspace name.		a node in tagPath in repository.
getAllTagStyle(String repository, String workspace) throws Exception ;	`repository`: The repository name `workspace`: The workspace name.	List<Node>	Get all tag style bases of a folksonomy tree.
init(String repository) throws Exception ;	`repository`: The repository name.	void	Initialize all TagStylePlugin with session in repository name.
removeTagOfDocument(String tagPath, Node document, String repository, String workspace) throws Exception;	`tagName`: The name of a folksonomy tree. `tagName`: The name of a tag. `document`: The document which is added a link to tagName. `repository`: The repository name.	void	Remove a tag of a given document.
removeTag(String tagPath, String repository, String workspace) throws Exception;	`tagPath`: The path of the tag. `repository`: The repository name. `workspace`: The workspace name.	void	Remove a tag.

Method	Return	Prototype	Description
modifyTagName(String tagPath, String newTagName, String repository, String workspace) throws Exception;	`tagPath`: The name of the tag. `newTagName`: The new tag name. `repository`: The repository name. `workspace`: The workspace name.	Node	Modify the tag name.

getLinkedTagsOfDocument(Node documentNode, String repository, String workspace) throws Exception;	`documentNode`: The document node.
	
	`repository`
	
	`workspace`
		List<Node>	Get all tags linked to a given document.

getLinkedTagsOfDocumentByScope(int scope, String value, Node documentNode, String repository, String workspace) throws Exception;	`documentNode`: The document node. `repository`: The repository name. `workspace`: The workspace name.	List<Node>	Get all tags linked to a given document by scope.

removeTagsOfNodeRecursively(Node node, String repository, String workspace, String username, String groups) throws Exception;	`node`
	
	`repository`
	
	`workspace`
		void	Remove all tags linked to the child nodes of a given node.

addTagPermission(String usersOrGroups);	`usersOrGroups`	void	Add given users or groups to tagPermissionList.
removeTagPermission(String usersOrGroups);	`usersOrGroups`	void	Remove given users or groups from tagPermissionList.

Method	Return	Prototype	Description
getTagPermissionList()	N/A	List<String>	Return tagPermissionList.

canEditTag(int scope, List<String> memberships);	`scope`
	
	`memberships`
		boolean	Set the permission to edit a tag for a user.

getAllTagNames(String repository, String workspace, int scope, String value) throws Exception;	`repository`
	
	`workspace`
	
	`scope`: The scope of tags.
	
	`value`: The value, according to scope, can be understood differently.
		List<String>	Get all tag names which start within a given scope.

ApplicationTemplateManager

This class is used to manage dynamic groovy templates for WCM-based products.

Package org.exoplatform.services.cms.views.ApplicationTemplateManager;

Method	Param	Return	Description
addPlugin(PortletTemplatePlugin portletTemplatePlugin) throws Exception	`portletTemplatePlugin`	void	Add the plugin..
getAllManagedPortletNames(String repository) throws Exception	`repository`	List<String>	Retrieve all the portlet names that have dynamic groovy templates managed by service.

getTemplatesByApplication(String repository, String portletName, SessionProvider provider)throws Exception;	`repository`
	
	`portletName`
	
	`provider`: The provider.
		List<String>	Retrieve the templates node by application.

getTemplatesByCategory(String repository, String	`repository`	List<String>	Retrieve the templates node by category.

Method	Param	Return	Description

portletName, String category, SessionProvider sessionProvider) throws Exception;	`portletName`
	`category`
	`sessionProvider`		

getTemplateByName(String repository, String portletName, String category, String templateName, SessionProvider sessionProvider)throws Exception;	`repository`
	`portletName`
	`category`
	`templateName`
	`sessionProvider`
		Node	Retrieve the template by name.

getTemplateByPath(String repository, String templatePath, SessionProvider sessionProvider)throws Exception ;	`repository`
	`templatePath`
	`sessionProvider`
		Node	Get the template by path.

addTemplate(Node portletTemplateHome, PortletTemplateConfig config)throws Exception;	`portletTemplateHome` `config`	void	Add the template.

removeTemplate(String repository, String portletName, String catgory, String templateName, SessionProvider sessionProvider)throws Exception;	`repository`
	`portletName`
	`catgory`
	`templateName`
	`sessionProvider`
		void	Remove the template.

NodeFinder

NodeFinder is used to find a node with a given path. If the path to the node contains sub-paths to exo:symlink nodes, find the real link node.

Method	Param	Return	Description
getNode(Node ancestorNode, String relativePath) throws PathNotFoundException, RepositoryException;	`ancestorNode`: The ancestor of the node to retrieve from which we start. `relativePath`: The relative path of the node to retrieve.	Node	Return the node at relPath related to the ancestor node.
getNode(Node ancestorNode, String relativePath, boolean giveTarget) throws PathNotFoundException, RepositoryException;	`ancestorNode`: The ancestor of the node to retrieve from which we start. `relativePath`: The relative path of the node to retrieve. `giveTarget`: Indicate if the target must be returned in case the item is a link.	Node	Return the node at relPath related to the ancestor node. If the node is a link and giveTarget has been set to <code>true</code>, the target node will be returned.
getItem(String repository, String workspace, String absPath) throws PathNotFoundException, RepositoryException;	`repository`: The repository name. `workspace`: The workspace name. `absPath`: An absolute path.	Item	Return the item at the specified absolute path.
getItemSys(String repository, String workspace, String absPath, boolean system) throws PathNotFoundException, RepositoryException;	`repository`: The name of repository `workspace`: The workspace name. `absPath`: An absolute path.	Item	Return the item at the specified absolute path.

Method	Param	Return	Description
getItem(String repository, String workspace, String : absPath, boolean giveTarget) PathNotFoundException, RepositoryException;	`repository`: The repository name. `workspace`: The workspace name. `absPath`: An absolute path. `giveTarget`: Indicate if the target must be returned in case the item is a link.	Item	Return the item at the specified absolute path. If the item is a link and giveTarget has been set to <code>true</code>, the target node will be returned.
getItemGiveTargetSys(String repository, String workspace, String absPath, boolean giveTarget, boolean system) throws PathNotFoundException, RepositoryException;	`repository`: The repository name. `workspace`: The workspace name. `absPath`: An absolute path. `giveTarget`: Indicate if the target must be returned in case the item is a link. `system`: The system provider.	Item	Return the item at the specified absolute path. If the item is a link and `giveTarget` has been set to <code>true</code>, the target node will be returned.
getItem(Session session, String absPath) PathNotFoundException, RepositoryException;	`session`: The session is used to get the item. `absPath`: An absolute path.	Item	Return the item at the specified absolute path.
getItem(Session session, String absPath, boolean giveTarget) PathNotFoundException, RepositoryException;	`session`: The session is used to get the item. `absPath`: An absolute path.	Item	Return the item at the specified absolute path. If the item is a link and `giveTarget` has been

Method	Param	Return	Description
	`giveTarget`: Indicate if the target must be returned in case the item is a link.		set to <code>true</code>, the target node will be returned.
getItemTarget(Session session, String absPath, boolean giveTarget, boolean system) throws PathNotFoundException, RepositoryException;	`session`: The session is used to get the item. `absPath`: An absolute path. `giveTarget`: Indicate if the target must be returned in case the item is a link. `system`: The system provider	Item	Return the item at the specified absolute path. If the item is a link and `giveTarget` has been set to <code>true</code>, the target node will be returned.
itemExists(Session session, String absPath) throws RepositoryException;	`session`: The session is used to get the item. `absPath`: An absolute path.	boolean	Return <code>true</code> if an item exists at absPath; otherwise returns <code>false</code>. Also returns <code>false</code> if the specified absPath is malformed.

JodConverter

JodConverter is used to convert documents into different office formats.

Package org.exoplatform.services.cms.jodconverter.JodConverterService

Method	Param	Return	Description

convert(InputStream input, String formatInput, OutputStream out,	`input`
	
	`formatInput`
		`void`	Convert InputStream in the formatInput format to OutputStream with the formatOutput format.

Method	Param	Return	Description

String formatOutput) throws Exception;	`out`
	
	`formatOutput`		

SiteSearchService

SiteSearchService is used in the Search portlet that allows users to find all information matching with your given keyword.

Method	Param	Return	Description
addExcludeIncludeDataTypePlugin(ExcludeIncludeDataTypePlugin plugin)	plugin: The plugin	void	Filter mimetypes data in the search results.
searchSiteContents(SessionProvider sessionProvider, QueryCriteria queryCriteria, int pageSize, boolean isSearchContent) throws Exception;	queryCriteria: The query criteria for SiteSearchService. Basing on search criteria, SiteSearchService can easily create the query statement to search. `sessionProvider`: The session provider. `pageSize`: The number of search results on a page.	AbstractPageList<ResultNode>	all child nodes whose contents match with the given keyword. These nodes will be put in the list of search results.

SEOService

SEOService supplies APIs to manage SEO data of a page or a content. This service includes some major functions which enables you to add, store, get or remove the metadata of a page or a content.

Method	Param	Return	Description
storePageMetadata(PageMetadataModel metaModel, String portalName, boolean onContent) throws Exception	metaModel: The metadata of a page/ content stored.	void	Store the metadata of a page/content.

Method	Param	Return	Description
	`portalName`: The name of portal. `onContent`: Indicate whether the current page is the content page or the portal page.		
getMetadata(ArrayList<String> params, String pageReference) throws Exception	params: The parameters list of a content page. `pageReference`: The reference of the page.	PageMetadataModel	Return the metadata of a portal page or a content page.
getPageMetadata (String pageReference) throws Exception	`pageReference`: The reference of the page.	PageMetadataModel	Return the metadata of a portal page.
getContentMetadata(ArrayList<String> params) throws Exception	params: The parameters list of a content page.	PageMetadataModel	Return the metadata of a content page.
removePageMetadata(PageMetadataModel metaModel, String portalName, boolean onContent) throws Exception	`metaModel`: The metadata of a page/ content stored. `portalName`: The name of portal. `onContent`: Indicate whether the current page is the content page or the portal page.	void	Remove the metadata of a page.
getContentNode(String contentPath) throws Exception	`contentPath`: The content path.	Node	Return the content node by the content path.
getHash(String uri) throws Exception	`uri`: The page reference of the UUID of a node.	String	Create a key from the page reference or the UUID of the node.

Method	Param	Return	Description
getSitemap(String portalName) throws Exception	`portalName`: The portal name.	String	Return a sitemap's content of a specific portal.
getRobots(String portalName) throws Exception	`portalName`: The portal name.	String	Return Robots' content of a specific portal.
getRobotsIndexOptions() throws Exception	N/A	List<String>	Return a list of options (INDEX and NOINDEX) for robots to index.
getRobotsFollowOptions() throws Exception	N/A	List<String>	Return a list of options (FOLLOW and NOFOLLOW) for robots to follow.
getFrequencyOptions() throws Exception	N/A	List<String>	Return a list of options for frequency.

FAQ

How to deploy a workflow?

The ***addPlugin()*** function of **WorkflowServiceContainer** service is used to register a Business Process when a workflow is implemented. Thus, if you want to use a workflow, you are required to configure the workflow service to invoke the ***addPlugin()*** function by adding the ***external-component-plugins*** element to the configuration file.

You have to set values for the name and location of the workflow which you want to use. There are **TWO** ways to configure the location of the workflow.

Deploy a workflow inside a .war file

You can use "war:(FOLDER_PATH)" to configure which **.jar** files contain your workflow processes inside the ***.war*** file.

```
<external-component-plugins>
        <target-component>org.exoplatform.services.workflow.WorkflowServiceContainer</target-component>
  <component-plugin>
    <name>deploy.predefined.processes</name>
    <set-method>addPlugin</set-method>
```

```
  <type>org.exoplatform.services.workflow.PredefinedProcessesPlugin</type>
  <init-params>
   <object-param>
    <name>predefined.processes</name>
    <description>load of default business processes</description>
    <object type="org.exoplatform.services.workflow.ProcessesConfig">
     <field name="processLocation">
      <string>war:/conf/bp</string>
     </field>
     <field name="predefinedProcess">
      <collection type="java.util.HashSet">
       <value>
        <string>/exo-ecms-ext-workflow-bp-jbpm-content-2.1.1.jar</string>
       </value>
       <value>
        <string>/exo-ecms-ext-workflow-bp-jbpm-payraise-2.1.1.jar</string>
       </value>
       <value>
        <string>/exo-ecms-ext-workflow-bp-jbpm-holiday-2.1.1.jar</string>
       </value>
      </collection>
     </field>
    </object>
   </object-param>
  </init-params>
 </component-plugin>
</external-component-plugins>
```

Deploy a workflow inside a .jar file

You can use "classpath:" to configure which **.jar** files contain your workflow processes inside the ***.jar*** file.

```
<external-component-plugins>
        <target-component>org.exoplatform.services.workflow.WorkflowServiceContainer</target-component>
 <component-plugin>
  <name>deploy.predefined.processes</name>
  <set-method>addPlugin</set-method>
  <type>org.exoplatform.services.workflow.PredefinedProcessesPlugin</type>
  <init-params>
   <object-param>
```

```
      <name>predefined.processes</name>
      <description>load of default business processes</description>
      <object type="org.exoplatform.services.workflow.ProcessesConfig">
        <field name="processLocation">
          <string>classpath:</string>
        </field>
        <field name="predefinedProcess">
          <collection type="java.util.HashSet">
            <value>
              <string>/exo-ecms-ext-workflow-bp-jbpm-content-myworkflow.jar</string>
            </value>
          </collection>
        </field>
      </object>
    </object-param>
  </init-params>
 </component-plugin>
</external-component-plugins>
```

The notification message is displayed when you deploy a workflow on Jboss. If you use "classpath:" to register, you must put your workflow in the **.jar** files inside the ***gatein.ear/lib*** folder (instead of the **lib** folder) to make it work.

eXo Platform 3.0 - CMIS Developer Guide

Introduction

The eXo Platform provides CMIS support using the xCMIS project and the WCM Storage provider.

About CMIS

The CMIS standard aims at defining a common content management web services interface that can be applied in content repositories and bring about the interoperability across repositories. The formal specification of CMIS standard is approved by the Organization for the Advancement of Structured Information Standards (OASIS) technical committee who drives the development, convergence and adoption of global information society. With CMIS, enterprises now can deploy systems independently, and create specialized applications running over a variety of content management systems.

To see the advantages of content interoperability and the significance of CMIS as a whole, it is necessary to learn about mutual targets which caused the appearance of specification first.

1. Content integration: With CMIS, integrating content among various repositories, even those created by different vendors in a single application, becomes faster, simpler and more effective. CMIS makes it possible for customers to integrate content management systems into their key business processes across business departments and vendor implementations.
2. Access unification: CMIS enables different applications and manufacturers to be connected to a CMIS-enabled content repository simply. With CMIS, a business application's developer can focus on the application's business logic, rather than issues related to the compatibility or content migration.

About xCMIS

xCMIS project, which is initially contributed to the Open Source community by eXo Platform, is an Open Source implementation of the Content Management Interoperability Services (CMIS) specification. xCMIS supports all the features stated in the CMIS core definition and both REST AtomPub and Web Services (SOAP/WSDL) protocol bindings.

To learn more about xCMIS, visit:

- [*http://gazarenkov.blogspot.com/2010/01/xcmis1-cmis-in-nutshell.html*]
- [*http://code.google.com/p/xcmis/*]

About eXo CMIS

eXo CMIS is built on the top of xCMIS embedded in Platform to expose the WCM drives as the CMIS repositories. The CMIS features are implemented as a set of components deployed on the eXo Container using XML files to describe the service configuration.

Note

SOAP protocol binding is not implemented in eXo CMIS.

Figure: how eXo CMIS works

WCM drives exposure is implemented as WCM storage provider to xCMIS SPI. Storage provider uses mappings from WCM's *ManageDriveService* to actual JCR nodes. And *AtomPub* bindings makes WCM structure available via CMIS standard API.

CMIS specification

Note

This is related to Content Management Interoperability Services (CMIS) Version 1.0 OASIS Standard 1 May 2010

[http://en.wikipedia.org/wiki/Content_Management_Interoperability_Services]

The CMIS interface is designed to be layered on top of existing Content Management systems and their existing programmatic interfaces. It is intended to expose all of the CM systems capabilities through the CMIS interfaces exhaustively. The CMIS specification defines the followings:

- A standard "domain model" for an ECM system - a set of core concepts included in all modern ECM systems, such as Object Types, properties, folders, documents, versions, and relationships; and a set of operations performed on those concepts, such as updating documents, or navigating via a folder hierarchy.
- The way to bind the CMIS domain model to two different web service protocols, including the Simple Object Access Protocol (SOAP) used in many ECM systems, and the Atom used in many Web 2.0 applications.

Note

SOAP protocol is not implemented in eXo CMIS.

CMIS specification provides a Web services interface which can:

- Work over existing repositories, enabling customers to build and leverage applications against multiple repositories.
- Decouple Web services and content from the content management repository, enabling customers to manage content independently.
- Provide common Web services and Web 2.0 interfaces to dramatically simplify the application development.
- Build the development platform and language agnostic.
- Support the composite application development and mashups by the business or IT analysts.

xCMIS project

xCMIS includes the client side frameworks for integrating content from different enterprise repositories, according to *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=cmis* [CMIS standard].

The project is to make joining Enterprise Content repositories simpler by offering CMIS ablity and exposing them to language-independent CMIS clients via the most convenient protocol.

xCMIS project:

- Is embedded, packaged as the J2EE Web archive (WAR) and prepared "download and go" Tomcat bundle.
- Has a live demo with the full-featured CMIS Expert client, which is accessible on the xcmis.org site and with prepared "download and go" Tomcat bundle (the client is accessible as the remote gadget).
- Is embedded in Platform to create the special xCMIS jcr repository and access it with any CMIS client.
- Tested with third-party CMIS clients, such as IBM CMIS Firefox Connector and CMIS Spaces Flex+AIR client. Either local repository (as described here) or *http://xcmis.org/xcmis1/rest/cmisatom* can be used as a CMIS repository's endpoint URL for these or other types of clients.

Benefits of xCMIS

- xCMIS is an open source, server side Java CMIS implementation, enabling to expose content in the existing content repositories according to the protocols defined in the CMIS specification.
- xCMIS will give developers a way to make their content repositories "pluggable" on the server side based on the internal Storage Provider Interface and additional protocol on-demand bindings.
- xCMIS will provide (several) CMIS client frameworks for repository-application and repository-repository interactions. The programming language and supported protocol can be selected by users. For example, the reasonable choice for using web applications, gadgets, and/or mashups is JavaScript or GWT over REST AtomPub, while for inter-repository exchange, it may be Java over Web Services, i.e. WSDL/SOAP.)
- Both the server and client sides of xCMIS are easily integrated in the eXo Platform 3.0 infrastructure. In particular, xCMIS exposes the eXo JCR content repository and provides a framework for building web applications and gadgets for the GateIn portal.

The xCMIS project is distributed under the LGPL license. You can download sources on *http://code.google.com/p/xcmis/source/checkout* [Google code], or visit *http://code.google.com/p/xcmis/w/list* [Community Wiki] for more information.

CMIS features

The CMIS specification prescribes:

- Domain model
- Services

CMIS Domain Model

The CMIS Domain Model defines a repository as a container and an entry point to the objects that is quite simple and non-restrictive. The followings are some of the common entities of the domain model.

- Repository is a container of objects with a set of "capabilities" which may be different depending on implementation.
- Object is the entity managed by a CMIS repository.
- Object Type is a classification related to an object. It specifies a fixed and non-hierarchical set of properties ("schema") that all objects of that type have.
- Document Object is an elementary information entity.
- Folder Object is a collection of fileable objects.
- Relationship Object is used to describe a dependent object semantically.
- Policy Object represents an administrative policy applied to an object.
- Access Object defines permissions.
- Versioning is to support versioning for Document objects.
- Query is type-based in a simplified SQL SELECT statement.
- Change Log is a mechanism which enables applications to discover changes to the objects stored.

Note

CMIS specifies a query language based on the SQL-92 standard, plus the extensions, in conjunction with the model mapping defined by the CMIS's relational view.

All objects are classified by an Object Type which declares that all objects of the given type have some sets of properties in common. Each property consists of a set of attributes, such as the TypeID, the property ID, its display name, its query name, and more). There are only four base types, including Document, Folder, Relationship, and Policy. Also, you can extend those basic types of modifying a set of their properties.

Document Object and Folder Object are self-explanatory. Document Object has properties to hold document metadata, such as the document author, modification date and custom properties. It can also contain a content stream whether it is required, and renditions, such as a thumbnail view of document. Folder is used to contain objects. Apart from the default hierarchical structure, CMIS can optionally store objects in multiple folders or in no folders at all (so-called multifiling and unfiling capabilities). Relationship Object denotes the connection between two objects (target and source). An object can have multiple relationships with other objects. Policy Object is a way to define administrative policies in managing objects. For example, you can use a CMIS policy to define which documents are subject to retention policies.

CMIS Services

CMIS provides a set of services to access and manage the content or repository. These services include:

- Repository Services: To discover information about the repository and the object types defined for the repository.
- Navigation Services: To traverse the folder hierarchy in a CMIS repository, and to locate documents which are checked out.
- Object Services: To execute ID-based CRUD functions (Create, Retrieve, Update, Delete) on objects in a repository.
- Multi-filing Services (optional): To put an object in more than one folder (multi-filing) or outside the folder hierarchy (unfiling).
- Discovery Services: To search for queryable objects in a repository.
- Versioning Services: To check out, navigate to documents or update a Document Version Series (checkOut, cancelCheckOut, getPropertiesOfLatestVersion, getAllVersions, deleteAllVersions).
- Relationship Services (optional): To retrieve an object for its relationships.
- Policy Services (optional): To apply, remove, or query for policies.
- ACL Services: To return and manage the Access Control List (ACL) of an object. ACL Services are not supported by all repositories.

Some repositories might not implement certain optional capabilities, but they are still considered CMIS-compliant. Each service has binding which defines the way messages will be serialized and wired. Binding are based on HTTP and uses the Atom Publishing Protocol.

Integration with eXo WCM

eXo Web Content Management (WCM) system provides CMIS access to its content storage features:

- WCM drives

- Document files and folders
- Symlinks
- Categories

To expose WCM drives as CMIS repositories there is special extension of *CmisRegistry*. Read admin guide for configuration of *org.exoplatform.ecms.xcmis.sp.jcr.exo.DriveCmisRegistry* component.

Work with CMIS is based on reference documents returned by services. Each CMIS service returns response containing links to other services describing the Document or operations on it. In most cases a Document will be asked by its ID. Some services accepts a Document path.

Note

Notes for use cases: To access the eXo CMIS services from the client side, use the Curl tool [http://curl.haxx.se/download.html]. CMIS AtomPub binding which is based upon the Atom (RFC4287) and Atom Publishing Protocol (RFC5023) will be used.

SOAP binding is not implemented in the eXo Platform 3.x.

JCR namespaces and nodetypes

eXo CMIS uses special JCR namespaces *cmis* and *xcmis* internally.

To expose drives content following nodetypes supported:

- nt:file nodetype for representation of cmis:documents
- nt:folder for representation of cmis:folder

Since CMIS specification not allows to have more root types except of described above (cmis:documents and cmis:folder), those two (nt:file and nt:folder) are mapped to CMIS types.

There is two more node types which are used: cmis:policy and cmis:relationship which represent CMIS types with corresponded (see Services description for details).

Additionally nodetypes used in WCM are mapped as follow:

- nt:unstructured + extentions as cmis:folder
- exo:taxonomy + extentions as cmis:folder

In other words only nodetypes extending nt:file, nt:folder, nt:unstructured and exo:taxonomy will be exposed correctly via CMIS API.

WCM's nodetype exo:article isn't supported by eXo CMIS due to not compliant structure to nt:file.

WCM drives as CMIS Repositories

The WCM drive is used to expose as an isolated repository via the CMIS service. Operations on the repository will reflect the drive immediately.

> Working with CMIS repositories it's important understand that a repository reflects a WCM Drive, which is a sub-tree in JCR workspace. And two or more drives can be mapped to a same workspace or a sub-tree. As a result changes in one repository can affect others. Consult with WCM drives mappings to know actual location of a content you'll access or change.

Use Case: Browse Drives via *getRepository*

- Login to the website as a user with the developer role.
- Open **Group | Sites Explorer**, you can see the drives set of WCM, such as **Sites Management**, **DMS Administration**.
- Get the list of these WCM drives via CMIS using *Curl*, asking *getRepositories* service:

```
curl -o getrepos.xml -u root:gtn
 http://localhost:8080/rest/private/cmisatom/
```

The requested file (getrepos.xml) contains the set of Repositories in the AtomPub format. The root element represents the set of **workspaces** representing **WCM drives** related to resources, for example for **DMS Administration** the response will contains data like:

```
    <service  xmlns="http://www.w3.org/2007/app"  xmlns:atom="http://www.w3.org/2005/Atom"
xmlns:cmisra="http://docs.oasis-open.org/ns/cmis/restatom/200908/">
  <workspace>
    <atom:title type="text">DMS Administration</atom:title>
    <cmisra:repositoryInfo xmlns:cmis="http://docs.oasis-open.org/ns/cmis/core/200908/">
      <cmis:repositoryId>DMS Administration</cmis:repositoryId>
      <cmis:repositoryName>DMS Administration</cmis:repositoryName>
    </cmisra:repositoryInfo>
    <collection href="http://localhost:8080/rest/private/cmisatom/DMS%20Administration/query">
      <atom:title type="text">Query</atom:title>
      <cmisra:collectionType>query</cmisra:collectionType>
    </collection>
    <collection
                href="http://localhost:8080/rest/private/cmisatom/DMS%20Administration/children/
00exo0jcr0root0uuid0000000000000">
```

```
        <atom:title type="text">Folder Children</atom:title>
        <cmisra:collectionType>root</cmisra:collectionType>
    </collection>
            <collection   href="http://localhost:8080/rest/private/cmisatom/DMS%20Administration/
checkedout">
        <atom:title type="text">Checkedout collection</atom:title>
        <cmisra:collectionType>checkedout</cmisra:collectionType>
    </collection>
   <collection href="http://localhost:8080/rest/private/cmisatom/DMS%20Administration/unfiled">
        <atom:title type="text">Unfiled collection</atom:title>
        <cmisra:collectionType>unfiled</cmisra:collectionType>
    </collection>
    <collection href="http://localhost:8080/rest/private/cmisatom/DMS%20Administration/types">
        <atom:title type="text">Types Children</atom:title>
        <cmisra:collectionType>types</cmisra:collectionType>
    </collection>
    <cmisra:uritemplate>
            <cmisra:template>http://localhost:8080/rest/private/cmisatom/DMS%20Administration/
object/
{id}?filter={filter}&amp;includeAllowableActions={includeAllowableActions}&amp;includePolicyIds={includePolicyIds}
        </cmisra:template>
        <cmisra:type>objectbyid</cmisra:type>
        <cmisra:mediatype>application/atom+xml;type=entry</cmisra:mediatype>
    </cmisra:uritemplate>
    <cmisra:uritemplate>
                                    <cmisra:template>http://localhost:8080/rest/private/cmisatom/
DMS%20Administration/
objectbypath?path={path}&amp;filter={filter}&amp;includeAllowableActions={includeAllowableActions}&amp;include
        </cmisra:template>
        <cmisra:type>objectbypath</cmisra:type>
        <cmisra:mediatype>application/atom+xml;type=entry</cmisra:mediatype>
    </cmisra:uritemplate>
    <cmisra:uritemplate>
                                    <cmisra:template>http://localhost:8080/rest/private/cmisatom/
DMS%20Administration/
query?q={q}&amp;searchAllVersions={searchAllVersions}&amp;maxItems={maxItems}&amp;skipCount={skipCount
        </cmisra:template>
        <cmisra:type>query</cmisra:type>
        <cmisra:mediatype>application/atom+xml;type=feed</cmisra:mediatype>
    </cmisra:uritemplate>
    <cmisra:uritemplate>
            <cmisra:template>http://localhost:8080/rest/private/cmisatom/DMS%20Administration/
typebyid/{id}
        </cmisra:template>
```

```
        <cmisra:type>typebyid</cmisra:type>
        <cmisra:mediatype>application/atom+xml;type=entry</cmisra:mediatype>
      </cmisra:uritemplate>
    </workspace>
  </service>
```

Here are the collection of services and predefined templates which can be used from the client side to request resources related to this repository. For example, to get the WCM node of **DMS Administration** drive by path, the **objectbypath** template can be used:

```
http://localhost:8080/rest/private/cmisatom/DMS%20Administration/
objectbypath?path={path}&amp;amp;filter={filter}&amp;amp;includeAllowableActions={includeAllowableActions}&amp;amp;include
```

where parameters include:

- Required:
  - ID repositoryId: The identifier for the repository.
  - String path: The path to the object.
- Optional:
  - String filter
  - Boolean includeAllowableActions
  - Enum includeRelationships
  - String renditionFilter
  - Boolean includePolicyIds
  - Boolean includeACL

Note

Find full description of all sepcefied services in the CMIS specefication.

WCM Symlinks

Symlinks are used to organize the virtual access to documents in WCM, which is implemented like links in Unix/Linux/Mac OS (refer to ln command for more details).

Note

JCR exo:symlink nodetype used for such type nodes.

Use Case: Follow Symlinks

- Login to the acme website as a user with the developer role.
- Open **Group | Sites Explorer**, select **Sites Management**, go to folder /acme/documents.
- Upload any file (e.g. test.txt) /acme/documents
- Add this file to **acme/News** category. It will create symlink to /acme/documents/test.txt in /acme/categories/acme/News
- Get content of folder /acme/categories/acme/News via CMIS:

```
curl -o news.xml -u root:gtn
 http://localhost:8080/rest/private/cmisatom/Managed%20Sites/
objectbypath?path=/acme/categories/acme/News
```

The requested file (products.xml) contains the entry with information about the folder.

- The list of properties for the object (such as node Id or type).
- Allowable Actions can be performed on the document; for example, requesting the children list.
- ACL and policies information.

```
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:cmisra="http://docs.oasis-open.org/ns/cmis/
restatom/200908/">
  <id>f59a3539c0a80003625790bdadf566c5</id>
  <published>2010-09-09T18:11:57.707Z</published>
  <updated>2010-09-09T18:11:57.707Z</updated>
  <summary type="text"></summary>
  <author>
    <name>system</name>
  </author>
  <title type="text">News</title>
       <link  href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites"  rel="service"
 type="application/atomsvc+xml"></link>
              <link     href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/object/
f59a3539c0a80003625790bdadf566c5"
    rel="self"></link>
              <link     href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/object/
f59a3539c0a80003625790bdadf566c5"
    rel="edit"></link>
           <link    href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/typebyid/
exo%3Ataxonomy" rel="describedby"
```

```
    type="application/atom+xml; type=entry"></link>
  <link
            href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/allowableactions/
f59a3539c0a80003625790bdadf566c5"
          rel="http://docs.oasis-open.org/ns/cmis/link/200908/allowableactions"  type="application/
cmis+xml; type=allowableActions"></link>
  <link
                href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/relationships/
f59a3539c0a80003625790bdadf566c5"
              rel="http://docs.oasis-open.org/ns/cmis/link/200908/relationships"   type="application/
atom+xml; type=feed"></link>
  <link
                      href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/policies/
f59a3539c0a80003625790bdadf566c5"
        rel="http://docs.oasis-open.org/ns/cmis/link/200908/policies"  type="application/atom+xml;
 type=feed"></link>
              <link      href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/objacl/
f59a3539c0a80003625790bdadf566c5"
       rel="http://docs.oasis-open.org/ns/cmis/link/200908/acl"  type="application/cmisacl+xml"></
link>
  <link
                      href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/children/
f59a3539c0a80003625790bdadf566c5"
    rel="down" type="application/atom+xml; type=feed"></link>
  <link
                href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/descendants/
f59a3539c0a80003625790bdadf566c5"
    rel="down" type="application/cmistree+xml"></link>
  <link
                    href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/foldertree/
f59a3539c0a80003625790bdadf566c5"
      rel="http://docs.oasis-open.org/ns/cmis/link/200908/foldertree" type="application/atom+xml;
 type=feed"></link>
              <link      href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/object/
f59a3533c0a8000339af97059f243a25"
    rel="up" type="application/atom+xml; type=entry"></link>
  <content type="text">News</content>
                <cmisra:object       xmlns:cmis="http://docs.oasis-open.org/ns/cmis/core/200908/"
 xmlns:cmisra="http://docs.oasis-open.org/ns/cmis/restatom/200908/">
    <cmis:properties>
                        <cmis:propertyId    propertyDefinitionId="cmis:allowedChildObjectTypeIds"
 localName="cmis:allowedChildObjectTypeIds"
                                                displayName="cmis:allowedChildObjectTypeIds"
 queryName="cmis:allowedChildObjectTypeIds"></cmis:propertyId>
```

```
        <cmis:propertyString propertyDefinitionId="cmis:path" localName="cmis:path"
          displayName="cmis:path" queryName="cmis:path">
          <cmis:value>/acme/categories/acme/News</cmis:value>
        </cmis:propertyString>
        <cmis:propertyId propertyDefinitionId="cmis:objectTypeId" localName="cmis:objectTypeId"
          displayName="cmis:objectTypeId" queryName="cmis:objectTypeId">
          <cmis:value>exo:taxonomy</cmis:value>
        </cmis:propertyId>
                                <cmis:propertyString     propertyDefinitionId="cmis:lastModifiedBy"
 localName="cmis:lastModifiedBy"
                          displayName="cmis:lastModifiedBy"   queryName="cmis:lastModifiedBy"></
cmis:propertyString>
        <cmis:propertyString propertyDefinitionId="cmis:name" localName="cmis:name"
          displayName="cmis:name" queryName="cmis:name">
          <cmis:value>News</cmis:value>
        </cmis:propertyString>
        <cmis:propertyString propertyDefinitionId="cmis:createdBy" localName="cmis:createdBy"
          displayName="cmis:createdBy" queryName="cmis:createdBy"></cmis:propertyString>
        <cmis:propertyId propertyDefinitionId="cmis:objectId" localName="cmis:objectId"
          displayName="cmis:objectId" queryName="cmis:objectId">
          <cmis:value>f59a3539c0a80003625790bdadf566c5</cmis:value>
        </cmis:propertyId>
                                <cmis:propertyDateTime     propertyDefinitionId="cmis:creationDate"
 localName="cmis:creationDate"
                            displayName="cmis:creationDate"   queryName="cmis:creationDate"></
cmis:propertyDateTime>
                                  <cmis:propertyString     propertyDefinitionId="cmis:changeToken"
 localName="cmis:changeToken"
                           displayName="cmis:changeToken"   queryName="cmis:changeToken"></
cmis:propertyString>
        <cmis:propertyId propertyDefinitionId="cmis:baseTypeId" localName="cmis:baseTypeId"
          displayName="cmis:baseTypeId" queryName="cmis:baseTypeId">
          <cmis:value>cmis:folder</cmis:value>
        </cmis:propertyId>
        <cmis:propertyId propertyDefinitionId="cmis:parentId" localName="cmis:parentId"
          displayName="cmis:parentId" queryName="cmis:parentId">
          <cmis:value>f59a3533c0a8000339af97059f243a25</cmis:value>
        </cmis:propertyId>
        <cmis:propertyDateTime propertyDefinitionId="cmis:lastModificationDate"
                  localName="cmis:lastModificationDate"  displayName="cmis:lastModificationDate"
 queryName="cmis:lastModificationDate"></cmis:propertyDateTime>
      </cmis:properties>
      <cmis:acl></cmis:acl>
      <cmis:exactACL>false</cmis:exactACL>
```

```
    <cmis:policyIds></cmis:policyIds>
    <cmis:rendition></cmis:rendition>
  </cmisra:object>
</entry>
```

To get the file which has uploaded above, use the **children** service to get the list of child nodes of /acme/documents. Find this service URL in the response XML above:

```
curl -o childs.xml -u root:gtn
 http://localhost:8080/rest/private/cmisatom/Managed%20Sites/children/
f59a3539c0a80003625790bdadf566c5
```

In the requested file (childs.xml), find the entry related to **test.txt** file uploaded via **Sites Explorer**. Find by title for name "test.txt".

```
<entry xmlns:cmisra="http://docs.oasis-open.org/ns/cmis/restatom/200908/">
  <id>f708e208c0a80003554babb97bd934ba</id>
  <published>2010-09-09T18:06:31.987Z</published>
  <updated>2010-09-09T18:06:31.987Z</updated>
  <summary type="text"></summary>
  <author>
    <name>system</name>
  </author>
  <title type="text">test.txt</title>
  <link href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites" rel="service"
    type="application/atomsvc+xml"></link>
  <link
                href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/object/
f708e208c0a80003554babb97bd934ba"
    rel="self"></link>
  <link
                href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/object/
f708e208c0a80003554babb97bd934ba"
    rel="edit"></link>
        <link  href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/typebyid/
cmis%3Adocument"
    rel="describedby" type="application/atom+xml; type=entry"></link>
  <link
        href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/allowableactions/
f708e208c0a80003554babb97bd934ba"
      rel="http://docs.oasis-open.org/ns/cmis/link/200908/allowableactions" type="application/
cmis+xml; type=allowableActions"></link>
```

```
    <link
                href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/relationships/
f708e208c0a80003554babb97bd934ba"
              rel="http://docs.oasis-open.org/ns/cmis/link/200908/relationships"  type="application/
atom+xml; type=feed"></link>
    <link
                    href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/policies/
f708e208c0a80003554babb97bd934ba"
       rel="http://docs.oasis-open.org/ns/cmis/link/200908/policies" type="application/atom+xml;
 type=feed"></link>
    <link
                     href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/objacl/
f708e208c0a80003554babb97bd934ba"
      rel="http://docs.oasis-open.org/ns/cmis/link/200908/acl" type="application/cmisacl+xml"></
link>
    <link
                   href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/versions/
f708a0f1c0a8000333e3681f99fab760"
      rel="version-history" type="application/atom+xml; type=feed"></link>
    <link
                     href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/object/
f708e208c0a80003554babb97bd934ba?returnVersion=latest"
      rel="current-version" type="application/atom+xml; type=entry"></link>
    <link
                    href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/parents/
f708e208c0a80003554babb97bd934ba"
      rel="up" type="application/atom+xml; type=feed"></link>
    <link
                          href="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/file/
f708e208c0a80003554babb97bd934ba"
      rel="edit-media"></link>
    <content type="text/plain"
                           src="http://localhost:8080/rest/private/cmisatom/Managed%20Sites/file/
f708e208c0a80003554babb97bd934ba"></content>
                  <cmisra:object    xmlns:cmis="http://docs.oasis-open.org/ns/cmis/core/200908/"
 xmlns:cmisra="http://docs.oasis-open.org/ns/cmis/restatom/200908/">
      <cmis:properties>
        <cmis:propertyBoolean propertyDefinitionId="cmis:isLatestMajorVersion"
            localName="cmis:isLatestMajorVersion" displayName="cmis:isLatestMajorVersion"
 queryName="cmis:isLatestMajorVersion">
          <cmis:value>false</cmis:value>
        </cmis:propertyBoolean>
        <cmis:propertyInteger propertyDefinitionId="cmis:contentStreamLength"
```

```
            localName="cmis:contentStreamLength" displayName="cmis:contentStreamLength"
queryName="cmis:contentStreamLength">
          <cmis:value>38</cmis:value>
        </cmis:propertyInteger>
                                <cmis:propertyId    propertyDefinitionId="cmis:contentStreamId"
localName="cmis:contentStreamId"
          displayName="cmis:contentStreamId" queryName="cmis:contentStreamId">
          <cmis:value>f708a0c2c0a800033bedeb35caddeed1</cmis:value>
        </cmis:propertyId>
      <cmis:propertyId propertyDefinitionId="cmis:objectTypeId" localName="cmis:objectTypeId"
          displayName="cmis:objectTypeId" queryName="cmis:objectTypeId">
          <cmis:value>cmis:document</cmis:value>
        </cmis:propertyId>
        <cmis:propertyString propertyDefinitionId="cmis:versionSeriesCheckedOutBy"
                                                localName="cmis:versionSeriesCheckedOutBy"
displayName="cmis:versionSeriesCheckedOutBy"
          queryName="cmis:versionSeriesCheckedOutBy"></cmis:propertyString>
                      <cmis:propertyId   propertyDefinitionId="cmis:versionSeriesCheckedOutId"
localName="cmis:versionSeriesCheckedOutId"
                                              displayName="cmis:versionSeriesCheckedOutId"
queryName="cmis:versionSeriesCheckedOutId"></cmis:propertyId>
        <cmis:propertyString propertyDefinitionId="cmis:name" localName="cmis:name"
          displayName="cmis:name" queryName="cmis:name">
          <cmis:value>test.txt</cmis:value>
        </cmis:propertyString>
        <cmis:propertyString propertyDefinitionId="cmis:contentStreamMimeType"
                                                  localName="cmis:contentStreamMimeType"
displayName="cmis:contentStreamMimeType" queryName="cmis:contentStreamMimeType">
          <cmis:value>text/plain</cmis:value>
        </cmis:propertyString>
                                  <cmis:propertyId    propertyDefinitionId="cmis:versionSeriesId"
localName="cmis:versionSeriesId"
          displayName="cmis:versionSeriesId" queryName="cmis:versionSeriesId">
          <cmis:value>f708a0f1c0a8000333e3681f99fab760</cmis:value>
        </cmis:propertyId>
                              <cmis:propertyDateTime    propertyDefinitionId="cmis:creationDate"
localName="cmis:creationDate"
          displayName="cmis:creationDate" queryName="cmis:creationDate">
          <cmis:value>2010-09-09T18:06:31.987Z</cmis:value>
        </cmis:propertyDateTime>
                                <cmis:propertyString    propertyDefinitionId="cmis:changeToken"
localName="cmis:changeToken"
                        displayName="cmis:changeToken"  queryName="cmis:changeToken"></
cmis:propertyString>
```

```
                    <cmis:propertyBoolean propertyDefinitionId="cmis:isLatestVersion"
localName="cmis:isLatestVersion"
      displayName="cmis:isLatestVersion" queryName="cmis:isLatestVersion">
      <cmis:value>true</cmis:value>
    </cmis:propertyBoolean>
                      <cmis:propertyString propertyDefinitionId="cmis:versionLabel"
localName="cmis:versionLabel"
      displayName="cmis:versionLabel" queryName="cmis:versionLabel">
      <cmis:value>latest</cmis:value>
    </cmis:propertyString>
    <cmis:propertyBoolean propertyDefinitionId="cmis:isVersionSeriesCheckedOut"
                                        localName="cmis:isVersionSeriesCheckedOut"
displayName="cmis:isVersionSeriesCheckedOut"
      queryName="cmis:isVersionSeriesCheckedOut">
      <cmis:value>false</cmis:value>
    </cmis:propertyBoolean>
                    <cmis:propertyString propertyDefinitionId="cmis:lastModifiedBy"
localName="cmis:lastModifiedBy"
                displayName="cmis:lastModifiedBy" queryName="cmis:lastModifiedBy"></
cmis:propertyString>
    <cmis:propertyString propertyDefinitionId="cmis:createdBy" localName="cmis:createdBy"
      displayName="cmis:createdBy" queryName="cmis:createdBy"></cmis:propertyString>
                    <cmis:propertyString propertyDefinitionId="cmis:checkinComment"
localName="cmis:checkinComment"
              displayName="cmis:checkinComment" queryName="cmis:checkinComment"></
cmis:propertyString>
    <cmis:propertyId propertyDefinitionId="cmis:objectId" localName="cmis:objectId"
      displayName="cmis:objectId" queryName="cmis:objectId">
      <cmis:value>f708e208c0a80003554babb97bd934ba</cmis:value>
    </cmis:propertyId>
                      <cmis:propertyBoolean propertyDefinitionId="cmis:isImmutable"
localName="cmis:isImmutable"
      displayName="cmis:isImmutable" queryName="cmis:isImmutable">
      <cmis:value>false</cmis:value>
    </cmis:propertyBoolean>
                    <cmis:propertyBoolean propertyDefinitionId="cmis:isMajorVersion"
localName="cmis:isMajorVersion"
      displayName="cmis:isMajorVersion" queryName="cmis:isMajorVersion">
      <cmis:value>false</cmis:value>
    </cmis:propertyBoolean>
    <cmis:propertyId propertyDefinitionId="cmis:baseTypeId" localName="cmis:baseTypeId"
      displayName="cmis:baseTypeId" queryName="cmis:baseTypeId">
      <cmis:value>cmis:document</cmis:value>
    </cmis:propertyId>
```

```
        <cmis:propertyString propertyDefinitionId="cmis:contentStreamFileName"
       localName="cmis:contentStreamFileName" displayName="cmis:contentStreamFileName"
queryName="cmis:contentStreamFileName">
          <cmis:value>test.txt</cmis:value>
        </cmis:propertyString>
        <cmis:propertyDateTime propertyDefinitionId="cmis:lastModificationDate"
              localName="cmis:lastModificationDate" displayName="cmis:lastModificationDate"
queryName="cmis:lastModificationDate">
          <cmis:value>2010-09-09T18:06:31.987Z</cmis:value>
        </cmis:propertyDateTime>
      </cmis:properties>
      <cmis:acl></cmis:acl>
      <cmis:exactACL>false</cmis:exactACL>
      <cmis:policyIds></cmis:policyIds>
      <cmis:rendition></cmis:rendition>
    </cmisra:object>
  </entry>
```

Then, get the **test.txt** file content via CMIS by using the **file** service and **id** of the file "test.txt" from **childs.xml**:

```
curl -o test.txt -u root:gtn
 http://localhost:8080/rest/private/cmisatom/Managed%20Sites/file/
f708e208c0a80003554babb97bd934ba
```

Get results in the test.txt file in the local folder. In this way we got file stored in **Sites Explorer** to folder /acme/documents/test.txt via **eXo CMIS** by symlink path /acme/categories/acme/News/test.txt. As it seen file's actual content referenced by id in CMIS. Path has no matter for content read or change.

Modify WCM via CMIS

Note

Upload the modified local file using the command with id of the file stored in WCM (it's jcr:uuid of the file in JCR Workspace). Note that you should run this command from the folder where the local file is stored.

To update any file via CMIS, use the **file** service and the PUT request. Use the same file as in the previous usecase (/acme/documents/test.txt, id:f708e208c0a80003554babb97bd934ba).

```
curl -T test.txt -X PUT -H "Content-Type:text/plain;
 charset=UTF-8" -u root:gtn
```

```
 http://localhost:8080/rest/private/cmisatom/Managed%20Sites/file/
f708e208c0a80003554babb97bd934ba
```

Go to the **Sites Explorer** and see changes applied from the local file in /acme/documents/test.txt.

CMIS search

CMIS provides a type-based query service for discovering objects that match specified criteria, by defining a read-only projection of the CMIS data model into a Relational View.

CMIS query languages is based on a subset of the SQL-92 grammar. CMIS-specific language extensions to SQL-92 are called out explicitly. The basic structure of a CMIS query is a SQL statement that MUST include the following clauses:

- SELECT (virtual columns): This clause identifies the set of virtual columns that will be included in the query results for each row.
- FROM (Virtual Table Names): This clause identifies which Virtual Table(s) the query will run against.

Additionally, a CMIS query MAY include the following clauses:

- WHERE (conditions): This clause identifies the constraints that rows MUST satisfy to be considered a result for the query.
- ORDER BY (sort specification): This clause identifies the order in which the result rows MUST be sorted in the result row set.

Each CMIS ObjectType definition has next query attributes:

- query name (String) - Used for query operations on object types. In our SQL statement examples all objecttypes is a queryName, i.e. the given queryName matches the specific type of document. For example, in query like "SELECT * FROM cmis:document" ,"cmis:document" is queryName preconfigured in Document object type definition.
- queryable (Boolean) - Indicates whether or not this object type is queryable. A non-queryable object type is not visible through the relational view that is used for query, and can not appear in the FROM clause of a query statement.
- fulltextIndexed (Boolean) - Indicates whether objects of this type are full-text indexed for querying via the CONTAINS() query predicate.
- includedInSupertypeQuery (Boolean) - Indicates whether this type and its subtypes appear in a query of this type's ancestor types. For example: if Invoice is a sub-type of Document, if this is TRUE on Invoice then for a query on Document type, instances of Invoice will be returned if they match. If this attribute is FALSE, no instances of Invoice will be returned even if they match the query.

Property definition also contains queryName and queriable attributes with same usage.

Query examples

Simple query

Query : Select all cmis:document.

```
SELECT * FROM cmis:document
```

Query result:

- All documents from Apollo program.

Find document by several constraints

Query : Select all documents where apollo:propertyBooster is 'Saturn V' and apollo:propertyCommander is Frank F. Borman, II or James A. Lovell, Jr.

Initial data:

- document1: apollo:propertyBooster - Saturn 1B, apollo:propertyCommander - Walter M. Schirra
- document2: apollo:propertyBooster - Saturn V, apollo:propertyCommander - Frank F. Borman, II
- document3: apollo:propertyBooster - Saturn V, apollo:propertyCommander - James A. Lovell, Jr.

```
SELECT * FROM cmis:document WHERE apollo:propertyBooster = 'Saturn V' AND (apollo:propertyCommander = 'Frank F. Borman, II' OR  apollo:propertyCommander = 'James A. Lovell, Jr.')
```

Query result:

- document2 and document3.

Full-text search

Query : Select all documents that contains "here" word.

Initial data:

- document1: content - "There must be test word"
- document2: content - "Test word is not here"

```
SELECT * FROM cmis:document WHERE CONTAINS('here')
```

Query result:

- document2.

Extended full-text search

Query: Select all documents that contains "There must" phrase and do not contain "check-word" word.

Initial data:

- document1: content - "There must be test word."
- document2: content - "Test word is not here. Another check-word."
- document3: content - "There must be check-word."

```
SELECT * FROM cmis:document WHERE CONTAINS("\"There must\" -\"check\-word\"")
```

Query result:

- document1.

Date property comparison

Query: Select all documents where cmis:lastModificationDate more than 2007-01-01.

Initial data:

- document1: cmis:lastModificationDate - 2006-08-08
- document2: cmis:lastModificationDate - 2009-08-08

```

```

```
SELECT * FROM cmis:document WHERE (cmis:lastModificationDate >= TIMESTAMP '2007-01-01T00:00:00.000Z')"
```

Query result:

- document2.

Boolean property comparison

Query: Select all documents where property apollo:someProperty equals to false.

Initial data:

- document1: apollo:someProperty - true
- document2: apollo:someProperty - false

```
SELECT * FROM cmis:document WHERE (apollo:someProperty = FALSE)
```

Query result:

- document2.

IN Constraint

Query : Select all documents where apollo:propertyCommander is in set {'Virgil I. Grissom', 'Frank F. Borman, II', 'James A. Lovell, Jr.'}.

Initial data:

- document1: apollo:propertyCommander - Walter M. Schirra
- document2: apollo:propertyCommander - Frank F. Borman, II
- document3: apollo:propertyCommander - James A. Lovell, Jr.
- document4: apollo:propertyCommander - Eugene A. Cernan

```
SELECT * FROM cmis:document WHERE apollo:propertyCommander IN ('Virgil I. Grissom', 'Frank F. Borman, II', 'James A. Lovell, Jr.')
```

Query result:

- document2 and document3.

Select all documents where longprop property NOT IN set

Query : Select all documents where apollo:propertyCommander property not in set {'Walter M. Schirra', 'James A. Lovell, Jr.'}.

Initial data:

- document1: apollo:propertyCommander - Walter M. Schirra
- document2: apollo:propertyCommander - Frank F. Borman, II
- document3: apollo:propertyCommander - James A. Lovell, Jr.
- document4: apollo:propertyCommander - Eugene A. Cerna

```
SELECT * FROM cmis:document WHERE apollo:PropertyCommander NOT IN ('Walter M. Schirra', 'James A. Lovell, Jr.')
```

Query result:

- document2,document4.

Select all documents where longprop property NOT NOT IN set

Query: Select all documents where apollo:propertyCommander property NOT IN set {'James A. Lovell, Jr.'}.

Initial data:

- document1: apollo:propertyCommander - Walter M. Schirra
- document2: apollo:propertyCommander - Frank F. Borman, II
- document3: apollo:propertyCommander - James A. Lovell, Jr.
- document4: apollo:propertyCommander - Eugene A. Cerna

```
SELECT * FROM cmis:document WHERE  NOT (apollo:propertyCommander NOT IN ('James A. Lovell, Jr.'))
```

Query result:

- document3.

IN_FOLDER constarint

Query: Select all folders that are in folder1.

Initial data:

- folder1: id - 123456789
  - document1: Title - node1
- folder3:
  - folder4:
- folder2:
  - document2: Title - node2

```
SELECT * FROM cmis:folder WHERE IN_FOLDER('123456789')
```

Query result:

- folder3.

Select all documents that are in specified folder

Query: Select all documents that are in folder1.

Initial data:

- folder1: id - 123456789
  - document1: Title - node1
- folder2:
  - document2: Title - node2

```
SELECT * FROM cmis:document WHERE IN_FOLDER('123456789')
```

Query result:

- document1.

Select all documents where query supertype is cmis:article

Initial data:

- testRoot: id - 123456789
- document1: Title - node1 typeID - cmis:article-sports
- document2: Title - node2 typeID - cmis:article-animals

```
SELECT * FROM cmis:article WHERE IN_FOLDER('123456789')
```

Query result:

- document1,document2.

IN_TREE constraint

Query: Select all documents that are in tree of folder1.

Initial data:

- folder1: id - 123456789
  - document1
- folder2:
  - document2

```
SELECT * FROM cmis:document WHERE IN_TREE('123456789')
```

Query result:

- document1,document2.

LIKE Comparison

Query: Select all documents where apollo:propertyCommander begins with "James".

Initial data:

- document1: apollo:propertyCommander - Walter M. Schirra
- document2: apollo:propertyCommander - Frank F. James, II
- document3: apollo:propertyCommander - James A. Lovell, Jr.
- document4: apollo:propertyCommander - Eugene A. James

```
SELECT * FROM cmis:document AS doc WHERE apollo:PropertyCommander LIKE 'James%'
```

Query result:

- document3.

Test LIKE constraint with escape symbols

Query : Select all documents where apollo:someProperty like 'admin%'.

Initial data:

- document1: Title - node1, apollo:someProperty - ad%min master
- document2: Title - node2, apollo:someProperty - admin operator
- document3: Title - node2, apollo:someProperty - radmin

```
SELECT * FROM cmis:document AS doc WHERE apollo:someProperty  LIKE 'ad\%min%'
```

Query result:

- document1.

NOT constraint

Query: Select all documents that not contains "world" word.

Initial data:

- document1: Title - node1, content - hello world
- document2: Title - node2, content - hello

```

```

```
SELECT * FROM cmis:document WHERE NOT CONTAINS('world')
```

Query result:

- document2.

Property existence

Query: Select all documents that has apollo:propertyCommander property IS NOT NULL.

Initial data:

- document1: apollo:propertyCommander - Walter M. Schirra
- document2: apollo:propertyCommander -
- document3: apollo:propertyCommander - James A. Lovell, Jr.
- document4: apollo:propertyCommander -

```
SELECT * FROM cmis:document WHERE apollo:propertyCommander IS NOT NULL
```

Query result:

- document1,document3.

ORDER BY

Query: Select all documents in default order (by document name).

Initial data:

- document1: Title - Apollo 7
- document2: Title - Apollo 8
- document3: Title - Apollo 13
- document4: Title - Apollo 17

```
SELECT cmis:lastModifiedBy, cmis:objectId, cmis:lastModificationDate FROM cmis:document
```

Query result:

- document3,document4,document1,document2.

ORDER BY ASC

Query: Order by apollo:propertyCommander property value (in ascending order).

Initial data:

- document1: apollo:propertyCommander - Walter M. Schirra
- document2: apollo:propertyCommander - Frank F. Borman, II
- document3: apollo:propertyCommander - James A. Lovell, Jr.
- document4: apollo:propertyCommander - Eugene A. Cerna

```
SELECT cmis:lastModifiedBy, cmis:objectId, cmis:lastModificationDate FROM cmis:document ORDER BY apollo:propertyCommander
```

Query result:

- document4,document2,document3,document1.

ORDER BY DESC

Query: Order by apollo:propertyCommander property value (in decending order).

Initial data:

- document1: apollo:propertyCommander - Walter M. Schirra
- document2: apollo:propertyCommander - Frank F. James, II
- document3: apollo:propertyCommander - James A. Lovell, Jr.
- document4: apollo:propertyCommander - Eugene A. James

```
SELECT cmis:lastModifiedBy, cmis:objectId, cmis:lastModificationDate FROM cmis:document ORDER BY cmis:propertyCommander DESC
```

Query result:

- document1,document3,document2,document4.

ORDER BY SCORE (as columns)

Query: Select all documents which contains word "moon" ordered by score.

Initial data:

- document1: content - "Earth-orbital mission, the first manned launch"
- document2: content - "from another celestial body - Earth's Moon"
- document3: content - "NASA intended to land on the Moon, but a mid-mission technical"
- document4: content - "It was the first night launch of a U.S. human"

```
SELECT cmis:lastModifiedBy, cmis:objectId, cmis:lastModificationDate FROM cmis:document WHERE CONTAINS('moon') ORDER BY SCORE()
```

Query result:

- document2,document3.

Not equal comparison (decimal)

Query: Select all documents property apollo:propertyBooster not equal to 3.

Initial data:

- document1: Title - node1, apollo:propertyBooster - 3
- document2: Title - node2, apollo:propertyBooster - 15

```
SELECT * FROM cmis:document WHERE apollo:propertyBooster <> 3
```

Query result:

- document2.

Not equal comparison (string)

Query: Select all documents property apollo:someProperty not equals to "test word second".

Initial data:

- document1: apollo:someProperty - "test word first"
- document2: apollo:someProperty - "test word second"

```
SELECT * FROM cmis:document WHERE apollo:someProperty <> 'test word second'
```

Query result:

- document1.

More than comparison (>)

Query: Select all documents property apollo:propertyBooster more than 5.

Initial data:

- document1: apollo:propertyBooster - 3
- document2: apollo:propertyBooster - 15

```
SELECT * FROM cmis:document WHERE apollo:propertyBooster > 5
```

Query result:

- document2.

CMIS Usage code examples

The following examples of CMIS usage may be useful for developers who needs to access a repository. CMIS access code snippets build using Apache HTTP Client for Java or using Google gadgets (gadgets.io) for JavaScript examples. For the cURL examples look at *http://code.google.com/p/xcmis/wiki/xCMISusesWithCurl.*

Login to repository

Note

CMIS service uses default authentication in general case, but it can be overridden in case of embedding CMIS into an Application Service. In these examples only Basic HTTP authentication covered.

Using java

```
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.UsernamePasswordCredentials;
import org.apache.commons.httpclient.auth.AuthScope;
import org.apache.commons.httpclient.methods.GetMethod;

HttpClient client = new HttpClient();
client.getState().setCredentials(
    new AuthScope("localhost", 8080, "realm"),
    new UsernamePasswordCredentials("root", "exo");
....
```

Listing of documents (folder, files)

There are a several methods to get the documents listings, such as getChildren(), getFolderTree() and getDescentants(), their usage will be described below. The difference between them is in usage of different URL segments to get a data ("/children" for getChildren(), "/foldertree" for getFolderTree(), "/descendants" for getDescentants()), and a different kind of results (getChildren() returns a flat structure, while a getFolderTree() and getDescentants() has a tree of items in response).

Using java

```
import org.apache.commons.httpclient.HttpClient;
```

```
import org.apache.commons.httpclient.methods.GetMethod;
import org.apache.commons.httpclient.MultiThreadedHttpConnectionManager;

String url = "http://localhost:8080/rest/private/cmisatom/";
url += repository;
url += "/children/";
url += obj_id;

HttpClient client = new HttpClient(new MultiThreadedHttpConnectionManager());
client.getHttpConnectionManager().
getParams().setConnectionTimeout(10000);

GetMethod get = new GetMethod(url);
try {
int result = client.executeMethod(get);
final String strResponse = get.getResponseBodyAsString();
} finally {
  get.releaseConnection();
}
```

Using JavaScript

Creating an URL to make a request (consists of repository name, the method name, e.g. "/children/", and folderID to get children from):

```
var url = "http://localhost:8080/rest/private/cmisatom/";
url += repository;
url += "/children/";
url += obj_id;
```

performing request:

```
var params = {};
params[gadgets.io.RequestParameters.METHOD] = gadgets.io.MethodType.GET;
params[gadgets.io.RequestParameters.CONTENT_TYPE] = gadgets.io.ContentType.FEED;
gadgets.io.makeRequest(url, handler, params);
```

processing results (code located in the handler specified while making a request, the same way it might be used for all examples in this chapter):

```
var handler = function(resp) {
  var data = eval(resp.data.Entry);
   for (var i = 0; i < data.length; i++) {
     var doc = data[i];
     alert(doc.Title);
     alert(doc.Date);
      ...etc..
   }
}
```

Read document properties and content-stream

Reading of Document properties and content stream it are two separate operations. Getting of content stream is possible after the properties set have been read and the content stream ID extracted from it.

Using java

Get document properties.

```
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.methods.GetMethod;
import org.apache.commons.httpclient.MultiThreadedHttpConnectionManager;

String url = "http://localhost:8080/rest/private/cmisatom/";
url += repository;
url += "/object/";
url += obj_id;

HttpClient client = new HttpClient(new MultiThreadedHttpConnectionManager());
client.getHttpConnectionManager().
getParams().setConnectionTimeout(10000);

GetMethod get = new GetMethod(url);
try {
  int result = client.executeMethod(get);
  final String strResponse = get.getResponseBodyAsString();
  // use response...
} finally {
  get.releaseConnection();
```

```
}
```

Get document content-stream.

To get a Document's content stream, an URL must contain "/file" part, object ID, and optionally the content stream ID, which can be used, for example, to obtain renditions. If no stream ID is specified, a default stream will be returned.

```
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.methods.PostMethod;

String url = "http://localhost:8080/rest/private/cmisatom/";
url += repository;
url += "/file/";
url += obj_id;
//Optionally
url += "?";
url += "streamid=";
url += streamID;

HttpClient client = new HttpClient();
client.getHttpConnectionManager().
getParams().setConnectionTimeout(10000);

GetMethod get = new GetMethod(url);
try {
  int result = client.executeMethod(get);
  final InputStream stream = get.getResponseBodyAsStream();
  try {
  // use stream...
  int dataByte = stream.read();
  } finally {
    stream.close();
  }
} finally {
  get.releaseConnection();
}
```

Using JavaScript

Get document properties.

Creating an URL to make a request (consists of repository name, method name, e.g. "/children/", and folder ID to get the children from):

```
var url = "http://localhost:8080/rest/private/cmisatom/";
url += repository;
url += "/object/";
url += obj_id;
```

performing request:

```
var params = {};
params[gadgets.io.RequestParameters.METHOD] = gadgets.io.MethodType.GET;
params[gadgets.io.RequestParameters.CONTENT_TYPE] = gadgets.io.ContentType.FEED;
gadgets.io.makeRequest(url, handler, params);
```

You can also use the ContentType.DOM parameter to parse the feed in your application (Using DOMParser for example).

Get document content-stream.

Note

Performing a content stream request in JavaScript will cause the browser dialog for a file download.

```
var url = "http://localhost:8080/rest/private/cmisatom/";
url += repository;
url += "/file/";
url += obj_id;
//Optionally
url += "?";
url += "streamid=";
url += streamID;
```

Search of data and syntax examples

CMIS supports SQL queries for more handful content search. Query service can handle both GET and POST requests. URL for query consists of repository name and method name "/query". GET

request must contain query as a parameter named "q", in case of POST request query must be located in a request body.

For more detailed instructions how to construct queries please refer to "Query examples" chapter.

Using java

```
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.methods.PostMethod;
import org.apache.commons.httpclient.methods.StringRequestEntity;

String url = "http://localhost:8080/rest/private/cmisatom/";
url += repository;
url += "/query/";

HttpClient client = new HttpClient();
client.getHttpConnectionManager().
getParams().setConnectionTimeout(10000);

PostMethod post = new PostMethod(url);
String s = "<?xml version='1.0' encoding='utf-8'?>"
            +  "<cmis:query  xmlns='http://www.w3.org/2005/Atom'  xmlns:cmis='http://docs.oasis-
open.org/ns/cmis/core/200908/'>"
       + "<cmis:statement>SELECT * FROM cmis:document</cmis:statement>"
       + "<cmis:maxItems>10</cmis:maxItems>"
       + "<cmis:skipCount>0</cmis:skipCount>"
       + "<cmis:searchAllVersions>true</cmis:searchAllVersions>"
       + "<cmis:includeAllowableActions>true</cmis:includeAllowableActions>"
       + "</cmis:query>";

RequestEntity entity = new StringRequestEntity(s, "text/xml","utf-8");
try {
  post.setRequestEntity(entity);
  int result = client.executeMethod(post);
  final String strResponse = post.getResponseBodyAsString();
  // use response...
} finally {
  post.releaseConnection();
}
```

Using JavaScript

```
var url = "http://localhost:8080/rest/private/cmisatom/";
url += repository;
url += "/query/";
```

```
var params = {};
params[gadgets.io.RequestParameters.METHOD] = gadgets.io.MethodType.POST;
params[gadgets.io.RequestParameters.POST_DATA] = gadgets.io.encodeValues(someQuery);
gadgets.io.makeRequest(url, handler, params);
```

Modification of document properties or content

Command of property update uses PUT method. The URL is the same as for a reading of properties, the difference is only in HTTP method used. Body of the request must be an Atom document with the properties specified (see spec. 2.2.4.12 for details constructing document).

Sending of content stream can be executed via PUT or POST requests. Content-type of the request must be an "multipart/form-data".

Using java

Update properties:

```
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.methods.StringRequestEntity;
import org.apache.commons.httpclient.methods.PostMethod;
import org.apache.commons.httpclient.methods.RequestEntity;

String url = "http://localhost:8080/rest/private/cmisatom/";
url += repository;
url += "/object/";
url += obj_id;

HttpClient client = new HttpClient();
client.getHttpConnectionManager().
getParams().setConnectionTimeout(10000);

String atomDoc = "<?xml version='1.0' encoding='utf-8'?>"
```

```
        + "<entry xmlns='http://www.w3.org/2005/Atom'"
        + " xmlns:cmis='http://docs.oasis-open.org/ns/cmis/core/200908/'"
        + " xmlns:cmisra='http://docs.oasis-open.org/ns/cmis/restatom/200908/'>"
        + "<cmisra:object><cmis:properties>"
                        + "<cmis:propertyString queryName='cmis:name' localName='cmis:name'
propertyDefinitionId='cmis:name'>"
        + "<cmis:value>newName</cmis:value>"
        + "</cmis:propertyString>"
        + "</cmis:properties></cmisra:object>"
        + "</entry>";

PutMethod put = new PutMethod(url);
RequestEntity entity = new StringRequestEntity(atomDoc, "text/xml", "utf-8");
put.setRequestEntity(entity);

try {
  int result = client.executeMethod(put);
  final String strResponse = put.getResponseBodyAsString();
} finally {
  put.releaseConnection();
}
```

Set content stream:

```
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.methods.InputStreamRequestEntity;
import org.apache.commons.httpclient.methods.PostMethod;
import org.apache.commons.httpclient.methods.RequestEntity;

String url = "http://localhost:8080/rest/private/cmisatom/";
url += repository;
url += "/file/";
url += obj_id;

HttpClient client = new HttpClient();
client.getHttpConnectionManager().
getParams().setConnectionTimeout(10000);

PostMethod post = new PostMethod(url);
 RequestEntity entity = new InputStreamRequestEntity(inputStream, "text/xml; charset=ISO-
8859-1");
post.setRequestEntity(entity);
```

```
try {
  int result = client.executeMethod(post);
  final String strResponse = post.getResponseBodyAsString();
} finally {
  post.releaseConnection();
}
```

Using JavaScript

Update properties:

```
var url = "http://localhost:8080/rest/private/cmisatom/";
url += repository;
url += "/object/";
url += obj_id;
```

```
//constructing document
 String atomDoc = "<?xml version='1.0' encoding='utf-8'?>";
      atomDoc += "<entry xmlns='http://www.w3.org/2005/Atom'";
      atomDoc += " xmlns:cmis='http://docs.oasis-open.org/ns/cmis/core/200908/'";
      atomDoc += " xmlns:cmisra='http://docs.oasis-open.org/ns/cmis/restatom/200908/'>";
      atomDoc += "<cmisra:object><cmis:properties>";
            atomDoc += "<cmis:propertyString queryName='cmis:name' localName='cmis:name'
 propertyDefinitionId='cmis:name'>";
      atomDoc += "<cmis:value>newName</cmis:value>";
      atomDoc += "</cmis:propertyString>";
      atomDoc += "</cmis:properties></cmisra:object></entry>";

 var params = {};
 params[gadgets.io.RequestParameters.METHOD] = gadgets.io.MethodType.PUT;
 params[gadgets.io.RequestParameters.POST_DATA] = atomDoc;
 gadgets.io.makeRequest(url, handler, params);
```

Set content stream:

```
var url = "http://localhost:8080/rest/private/cmisatom/";
```

```
url += repository;
url += "/file/";
url += obj_id;
```

```
var params = {};
params[gadgets.io.RequestParameters.METHOD] = gadgets.io.MethodType.POST;
params[gadgets.io.RequestParameters.CONTENT_TYPE] = "multipart/form-data";
params[gadgets.io.RequestParameters.POST_DATA] = contentStream;
gadgets.io.makeRequest(url, handler, params);
```

References

xCMIS project links:

- Community site *http://xcmis.org/*.
- JIRA site *https://jira.exoplatform.org/browse/CMIS*.
- Forum *http://groups.google.com/group/xcmis*.
- xCMIS latest news *http://gazarenkov.blogspot.com/*.
- eXo Platform blog *http://blog.exoplatform.org/*.
- *http://xcmis.org/CmisExpert1/org.exoplatform.cmis.CmisExpertApplication/CmisExpertApplication.html* [Demo site with CMISExpert client].

CMIS-related links:

- *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=cmis* [CMIS specification].
- TODO CMIS working group (list, mail, feeds etc).
- TODO CMIS external tutorials, guides, videos, blogs, news.
- TODO CMIS clients, UI, web.

eXo Platform 3.0 - CMIS Administrator Guide

Introduction

It is to introduce about basic knowledge of eXo CMIS, working principles and how to configure it.

CMIS Specification

Note

Content Management Interoperability Services (CMIS) Version 1.0 OASIS Standard 1 May 2010

[*http://en.wikipedia.org/wiki/Content_Management_Interoperability_Services*]

The Content Management Interoperability Services (CMIS) standard defines a domain model and Web Services and Restful AtomPub bindings that can be used by applications to work with one or more Content Management repositories/systems. The CMIS interface is designed to be layered on top of existing Content Management systems and their existing programmatic interfaces. It is not intended to prescribe how specific features should be implemented within those CM systems, nor to exhaustively expose all of the CM systems capabilities through the CMIS interfaces. Rather, it is intended to define a generic/universal set of capabilities provided by a CM system and a set of services for working with those capabilities.

The CMIS specification provides a Web services interface that:

- Is designed to work over existing repositories, enabling customers to build and leverage applications against multiple repositories.
- Decouples Web services and content from the content management repository, enabling customers to manage content independently.
- Provides common Web services and Web 2.0 interfaces to dramatically simplify application development.
- Is development platform and language agnostic.
- Supports the composite application development and mash-up by the business or IT analysts.

xCMIS project

The xCMIS project *http://xcmis.org/*, initially contributed to the Open Source community by *http://www.exoplatform.com* [eXo Platform], is an implementation of the full stack of Java-based CMIS services. xCMIS also includes the client side frameworks for integrating content from different enterprise repositories, according to the *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=cmis* [CMIS standard].

The project is distributed under the LGPL license. You can download sources at *http://code.google.com/p/xcmis/source/checkout* [Google code], or visit *http://code.google.com/p/xcmis/w/list* [Community Wiki] for more information.

eXo CMIS

eXo CMIS is one eXo Platform service exposing eXo Content via CMIS. eXo CMIS offers the way to access the eXo ECMS content from the CMIS 1.0 compatible Web client using the REST Atom protocol.

Note

WS-SOAP/WSDL binding is also supported, but it is not delivered in the standard eXo Platform 3.0 bundle.

eXo CMIS is built on the top of xCMIS embedded in Platform to expose the WCM drives as the CMIS repositories. The CMIS features are implemented as a set of components deployed on the eXo Container using XML files to describe the service configuration.

Figure: how eXo CMIS works

WCM drives exposure is implemented as WCM storage provider to xCMIS SPI. Storage provider uses mappings from WCM's *ManageDriveService* to actual JCR nodes. And *AtomPub* bindings makes WCM structure available via CMIS standard API.

Configuration

CMIS Configuration

To expose WCM drives to the CMIS repositories, you must make a special extension of *CmisRegistry*.

To make a typical component *org.exoplatform.ecms.xcmis.sp.jcr.exo.DriveCmisRegistry*, do as follows:

```
<component>
  <type>org.exoplatform.ecms.xcmis.sp.jcr.exo.DriveCmisRegistry</type>
  <init-params>
    <!-- Disabled by default. Uncomment if you need query support in CMIS. -->
      <!-- value-param>
        <name>indexDir</name>
        <value>${gatein.jcr.index.data.dir}/cmis-index${container.name.suffix}</value>
      </value-param-->
    <value-param>
      <name>exo.cmis.renditions.persistent</name>
      <value>true</value>
    </value-param>
    <values-param>
      <name>renditionProviders</name>
      <description>Redition providers classes.</description>
      <!-- <value>org.xcmis.renditions.impl.PDFDocumentRenditionProvider</value> -->
      <value>org.xcmis.renditions.impl.ImageRenditionProvider</value>
    </values-param>
  </init-params>
</component>
```

Where configuration parameters include:

- indexDir - directory where the lucene index will be placed. It is disabled by default.
- exo.cmis.renditions.persistent - Indicates if a rendition of the document (thumbnails) should be persisted in the JCR. The allowed value are *true* or *false*.
- renditionProviders - set of FQN of classes which can be used as Rendition Providers. Classes which implement the org.xcmis.spi.RenditionProvider interface are used to preview the CMIS objects (thumbnails).

Note

In most cases, it is not required to change anything in the xCIMIS configuration. In case of any change of the indexer storage location, do not comment the indexDir value parameter and point it to the actual location.

Required nodetypes and namespaces in JCR

The following configuration is mandatory for JCR to work correctly:

```
<external-component-plugins>
  <target-component>org.exoplatform.services.jcr.RepositoryService</target-component>
  <component-plugin>
    <name>add.namespaces</name>
    <set-method>addPlugin</set-method>
    <type>org.exoplatform.services.jcr.impl.AddNamespacesPlugin</type>
    <init-params>
      <properties-param>
        <name>namespaces</name>
        <property name="cmis" value="http://www.exoplatform.com/jcr/cmis/1.0" />
        <property name="xcmis" value="http://www.exoplatform.com/jcr/xcmis/1.0" />
      </properties-param>
    </init-params>
  </component-plugin>
  <component-plugin>
    <name>add.nodeType</name>
    <set-method>addPlugin</set-method>
    <type>org.exoplatform.services.jcr.impl.AddNodeTypePlugin</type>
    <init-params>
     <values-param>
        <name>autoCreatedInNewRepository</name>
        <description>Node types configuration file</description>
        <value>jar:/conf/cmis-nodetypes-config.xml</value>
      </values-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

Authenticator and organization service configuration

An Authenticator is responsible for creating Identity *Security Service Authenticator*

The eXo CMIS service is based on:

- The authentication mechanism provided by the eXo organization service.
- The JAAS configuration of eXo Platform. For example,

```
gatein-domain {
  org.exoplatform.web.security.PortalLoginModule required;
  org.exoplatform.services.security.jaas.SharedStateLoginModule required;
  org.exoplatform.services.security.j2ee.TomcatLoginModule required;
};
```

CMIS search and index

The CMIS standard defines a query language based on on a subset of the SQL-92 grammar (ISO/IEC 9075: 1992 -- Database Language SQL), with a few extensions to enhance its filtering capability for the CMIS data model, such as existential quantification for multi-valued property, full-text search, and folder membership.

> CMIS search is disabled by default in eXo CMIS. Uncomment indexDir parameter if you need query support in CMIS. To discover search capability check capabilityQuery property (see table below).

CMIS Relational View

The relational view of a CMIS repository consists of a collection of virtual tables that are defined on top of the CMIS data model. A virtual table exists for every queryable object type (content type if you prefer) in the repository. Each row in these virtual tables correspond to an instance of the corresponding object type (or of one of its subtypes). A column exists for every property that the object type has.

Query Capabilities

Table 2.1.

Capability	Value
capabilityQuery	`bothcombined` (if indexDir configured, otherwise `none`)
capabilityJoin	`none`
capabilityPWCSearchable	`false`
capabilityAllVersionsSearchable	`false`

Configuration

To be able to provide full-text search capabilities xCMIS uses its own index. There is configuration parameter:

Table 2.2.

Parameter	Default	Description
indexDir	none	The location of the index directory. This parameter is mandatory for default implementation

All these parameters can be passed through the xml configuration.

How to set up the index directory.

```
<component>
  <type>org.exoplatform.ecms.xcmis.sp.DriveCmisRegistry</type>
  <init-params>
      <!-- Disabled by default. Uncomment if you need query support in CMIS. -->
      <!-- value-param>
        <name>indexDir</name>
        <value>${gatein.jcr.index.data.dir}/cmis-index${container.name.suffix}</value>
      </value-param-->
      .............
  </init-params>
</component>
```

Index atomicity and durability

- Write-ahead logging

To be able to provide index consistency and recoverability in the case of unexpected crashes or damages, XCMIS uses *write-ahead logging* [http://en.wikipedia.org/wiki/Write-ahead_logging] (WAL) technique. Write-Ahead Logging is a standard approach to transaction logging. Briefly, WAL's central concept is that changes to data files (indexes) must be written only after those changes have been logged, that is, when log records describing the changes have been flushed to permanent storage. If we follow this procedure, we do not need to flush data pages to disk on every transaction commit, because we know that in the event of a crash we will be able to recover the index using the log: any changes that have not been applied to the data pages can be redone from the log records. (This is roll-forward recovery, also known as REDO.)

A major benefit of using WAL is a significantly reduced number of disk writes, because only the log file needs to be flushed to disk at the time of transaction commit, rather than every data file changed by the transaction.

- Recover uncommited transaction

When you start Indexer, it will check uncommitted transaction logs. If there are at least one log exists - recover process will be started. Indexer will read all logs and extract added, updated and removed uuids into set. Then it will walk throw this set and check objects according to UUID. If object exist - indexer will put into the added documents list, in other case uuid will be added to removed documents list. After according to the list of added and removed documents changes will be applyed to the index.

- Initial index population

When you run the indexer checks the number of documents in the index. If there is no documents in the index or previous re-indexation wasn't successful then re-indexation of all content will be started. First step is cleaning old index data. Uncommited transaction logs and old persistent data is removed. This data is useless, because re-indexation of all content will be started. Then indexer walks throw all objects and make lucene document for each one. Then batches with less then 100 elements will be saved to the index. After re-indexation, all logs (WAL) will be removed, all data mentioned on this changes logs - already indexed.

Note

If administrator get exception with message "Can't remove reindex flag." it means that restore index was finished but file-flag was not removed (see index directory, file named as "reindexProcessing"). You can manualy remove this file-flag, and avoid new reindex of repository on jcr start.

Service JARs

Note

JCRs listed for informational purpose only. E.g. if the customer application will need to use same third-parties used by eXo CMIS but of another version. All files are delivered in Platform 3.0 distribution.

eXo CMIS consists of JAR files:

xCMIS project files (actual for the xCMIS 1.2.x versions):

- xcmis-renditions-X.X.X.jar
- xcmis-restatom-X.X.X.jar
- xcmis-search-model-X.X.X.jar
- xcmis-search-parser-cmis-X.X.X.jar
- xcmis-search-service-X.X.X.jar
- xcmis-spi-X.X.X.jar

eXo WCM Storage Provider for xCMIS SPI (uses the same version as WCM/ECMS system):

- exo-ecms-ext-xcmis-sp-Y.Y.Y.jar

Third-party dependencies (of xCMIS 1.2.x):

- *abdera-client-0.4.0-incubating.jar*
- *abdera-core-0.4.0-incubating.jar*
- *abdera-i18n-0.4.0-incubating.jar*
- *abdera-parser-0.4.0-incubating.jar*
- *abdera-server-0.4.0-incubating.jar*
- *antlr-runtime-3.1.3.jar*
- *axiom-api-1.2.5.jar*
- *axiom-impl-1.2.5.jar*
- *jaxen-1.1.1.jar*
- *lucene-regex-2.4.1.jar*

etc.

Miscellaneous and Tips

Clients can be useful for CMIS repositories access:

- CMIS Expert, available at *http://www.xcmis.org* [xCMIS.org] as a Google gadget.

- IBM *CMIS Firefox Connector* [http://www.ibm.com/developerworks/lotus/library/quickr-cmis/index.html#N101CC]

  The IBM CMIS Firefox plugin is not compatible with Firefox 3.6.

- Flex+AIR and Flex+Browser CMIS clients, based on FlexSpaces framework. *http://code.google.com/p/cmisspaces*

Links

eXo CMIS links

xCMIS project links:

- Community site *http://xcmis.org/* with CMIS Expert demo
- Forum *http://groups.google.com/group/xcmis*
- xCMIS latest news *http://gazarenkov.blogspot.com/*
- eXo Platform blog *http://blog.exoplatform.org/*

CMIS related links:

- *http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=cmis* [CMIS specification]
- TODO CMIS working group (list, mail, feeds etc)
- TODO CMIS external tutorials, guides, videos, blogs, news
- CMIS clients
  - *CMIS Firefox Connector* [http://www.ibm.com/developerworks/lotus/library/quickr-cmis/index.html]
  - Flex+AIR and Flex+Browser CMIS clients *http://code.google.com/p/cmisspaces*

eXo Platform 3.0 - Collaboration Functions

Prerequisites

You can refer to the following links to understand more about the eXo Collaboration Reference Guide:

- *GateIn Reference Guide* [http://docs.jboss.com/gatein/portal/3.1.0-FINAL/reference-guide/en-US/html/index.html]
- *OpenSocial gadget* [http://wiki.opensocial.org/index.php?title=Main_Page]
- *WebService* [http://docs.jboss.org/exojcr/1.12.4-GA/developer/en-US/html/part-ws.html]
- *Chatserver* [http://www.igniterealtime.org/builds/openfire/docs/latest/documentation/javadoc/index.html]
- *Vcard* [http://en.wikipedia.org/wiki/VCard]
- *iCalendar* [http://en.wikipedia.org/wiki/ICalendar] and *its Internet standard* [http://www.ietf.org/rfc/rfc2445.txt]

Applications

Portlets

Calendar portlet

The Calendar portlet is packaged in the *calendar.war* file.

Description

The Calendar portlet shows the Calendar application of eXo Collaboration with a lot of features provided to users.

The Calendar application includes the following features:

- Create multiple personal calendars, manage calendars easily with calendar groups.
- Create events or tasks using the **Quick Add** dialog easily.
- Create events or tasks in details.
- Create all-day events.
- View other attender's availability schedules.
- Create recurring events.
- Get reminders.
- View calendars by various views: day, week, month and year.
- View events day-by-day by navigating the mini-calendar quickly.
- Share calendars with others.
- Import/Export calendars.
- Publish your calendars with RSS, CalDAV.
- Search for events and/or tasks in calendars.
- Print your agenda.

Portlet.xml

To see the portlet in the project, follow this path: */eXoApplication/calendar/webapp/src/main/webapp/WEB-INF/portlet.xml*.

Chatbar portlet

The Chatbar portlet is packaged in the *Chatbar.war* file.

Description

The Chatbar portlet shows the Chatbar application of eXo Collaboration that can be positioned in the portal or page layout as any other, but behaves as a floating box. The bar remains floating at its location even when the browser window is scrolled or resized. Its height is fixed, but can be expanded horizontally to any size available in its container. This allows the portlet to be placed in two layout cases:

- Large width area (typically header or footer).
- Narrow column.

The Chatbar application implements all functions of the Chat application, allowing you to send and receive messages anywhere after you are logged in successfully. The Chatbar is a typical toolbar with buttons to open menus. It gives access to main features of Chat:

- Status change and presence indicator.
- Contacts.
- Rooms.
- Minimized conversation window.

Portlet preferences

The Chatbar Portlet consists of some preferences as in the following sample code:

```
<portlet-preferences>
  <preference>
    <name>showMailLink</name>
    <value>true</value> <!--true/false -->
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showCalendarLink</name>
```

```
    <value>true</value> <!--true/false -->
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>showContactLink</name>
    <value>true</value> <!--true/false -->
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>mailUrl</name>
    <value>portal/private/intranet/mail</value> <!--String page name -->
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>calendarUrl</name>
    <value>portal/private/intranet/calendar</value> <!--String page name -->
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>contactUrl</name>
    <value>portal/private/intranet/contact</value> <!--String page name -->
    <read-only>false</read-only>
  </preference>
  <preference>
    <name>info</name>
       <value>info</value> <!--this is only the key to get the resource bundle the full key :
 UIConfigForm.label.info -->
    <read-only>true</read-only>
  </preference>
</portlet-preferences>
```

In which:

Preference Name	Possible Values	Default Values	Description
showMailLink	`true / false`	`true`	The value as "true" or "false" means that users are allowed to see the application icon or not respectively.
showCalendarLink	`true / false`	`true`	The value as "true" or "false" means that users are

Preference Name	Possible Values	Default Values	Description
			allowed to see the application icon or not respectively.
showContactLink	`true / false`	`true`	The value as "true" or "false" means that users are allowed to see the application icon or not respectively.
mailUrl	`string`	`Portal/private/intranet/mail`	The URL to the Mail application page in the portal without combining with the %domain name. The port% chatbar will resolve it from server.
calendarUrl	`string`	`Portal/private/intranet/calendar`	The URL to the Calendar application page in the portal without combining with the %domain name. The port% chatbar will resolve it from server.
contactUrl	`string`	`Portal/private/intranet/contact`	The URL to the Address Book application page in the portal without combining with the %domain name. The port% chatbar will resolve it from server.
info	`Info`	`Info`	This is only the key to get the resource bundle of the full key: UIConfigForm.label.info.

Portlet.xml

See the portlet in the project following this path: */eXoApplication/chatbar/webapp/src/main/webapp/WEB-INF/portlet.xml*.

Chat Portlet

The Chat portlet is packaged in the *Chat.war* file.

Description

The Chat Portlet shows the Chat application of eXo Collaboration that allows users to enter chat rooms and communicate with online others at real time.

Portlet.xml

See the portlet in the project following this path: */eXoApplication/chat/webapp/src/main/webapp/WEB-INF/portlet.xml*

Contact Portlet

Contact Portlet is packaged in the *Contact.war* file.

Description

Contact Portlet shows the Contact application of eXo Collaboration that allows users to personalize their contact view from different view types, such as List view and VCards view.

Portlet.xml

See the portlet in the project following this path: */eXoApplication/contact/webapp/src/main/webapp/WEB-INF/portlet.xml*.

Mail Portlet

The Mail Portlet is packaged in the *Mail.war* file.

Description

Mail Portlet shows the Mail application of eXo Collaboration that offers a lot of features to users such as sending, receiving or viewing their mails through Internet without actually downloading them to their computer. Users not only take advantages of eXo Mail by keeping and receiving all important messages, files and pictures forever but also by looking for and viewing their needed messages easily whenever they want. Additionally, the Mail application is smoothly integrated with other Collaboration modules, such as Address Book and Calendar.

Portlet.xml

See the portlet in the project following this path: */eXoApplication/mail/webapp/src/main/webapp/WEB-INF/portlet.xml*.

RSSreader Portlet

The RSSreader Portlet is packaged in the *Rssreader.war* file.

Description

eXo Collaboration uses the RSS Reader Portlet that facilitates users to quickly get a view of their favorite feeds around the web. They will get the latest news, the last updated posts from their favorite blogs, latest emails, and more.

Portlet.xml

See the portlet in the project following this path: */eXoApplication/content/webapp/src/main/webapp/WEB-INF/portlet.xml*.

Gadgets

eXo Collaboration consists of three gadgets: eventslist, taskslist and messageslist. They are packaged in the *csResources.war* file.

Eventslist

Description

Eventslist lists the maximum number of upcoming events, that is configurable by users. For example, they can set the preference list to 5 or 10 events.

See preferences of this gadget in the following sample code:

```
<UserPref datatype="string" display_name="__MSG_baseurl__" name="url" required="true" value="/calendar"/>
<UserPref datatype="string" display_name="__MSG_subscribeurl__" name="subscribeurl" required="true" value="/portal/rest/private/cs/calendar/upcoming"/>
<UserPref datatype="string" default_value="10" display_name="__MSG_limit__" name="limit"/>
<UserPref datatype="enum" default_value="AM/PM" display_name="__MSG_format__" name="timeformat"/>
```

Details:

Preferences	Description
url	Link to the Calendar portlet.
Subscribeurl	Link to the upcoming events.
limit	The maximum number of upcoming events.
timeformat	The time format for upcoming events.

For more details on the preferences of gadgets, see *here.* [http://code.google.com/apis/gadgets/docs/basic.html#Userprefs]

Links to used REST services

It uses the *upcomingEvent* service in the following package: *org.exoplatform.webservice.cs.calendar.CalendarWebservice.java*.

Taskslist

Taskslist lists the maximum number of upcoming tasks that is configurable by users. For example, they can set the preference list to 5 or 10 tasks.

Description

See the preferences of this gadget in the following sample code:

```
<UserPref datatype="hidden" default_value="/calendar:/portal/rest/private/cs/calendar/upcoming:10:AM/PM:Default" name="setting"/>
```

Accordingly, *setting* collects all the configuration of upcoming tasks and add some more functions to help developers change the configuration of the default skin.

Links to used REST services

It uses upcomingEvent service in the following package: *org.exoplatform.webservice.cs.calendar.CalendarWebservice.java*.

Messageslist

It lists the maximum number of unread messages, that is configurable by users.

Description

See the prefereces of this gadget in the following sample code:

```
<UserPref datatype="hidden" display_name="__MSG_baseurl__" name="url" required="true" value="/mail"/>
<UserPref datatype="hidden" display_name="__MSG_subscribeurl__" name="subscribeurl" required="true" value="/portal/rest/private/cs/mail/unreadMail"/>
<UserPref datatype="hidden" default_value="5" display_name="__MSG_limit__" name="limit"/>
<UserPref datatype="hidden" default_value="" display_name="__MSG_account__" name="account"/>
<UserPref datatype="hidden" default_value="" display_name="__MSG_folder__" name="folder"/>
```

```
<UserPref datatype="hidden" default_value="" display_name="__MSG_tag__" name="tag"/>
```

Details:

Preferences	Description
Url	The URL of the Mail Application.
Subscribeurl	The link to upcoming messages.
Limit	The number of displayed unread messages that is set by users.
Account	The mail account in the Mail application.
Folder	The folder which consists of unread messages.
Tag	The tags in all unread messages.

Links to used REST services

It uses the unreadMail service in the following package: *org.exoplatform.webservice.cs.mail.MailWebservice.java*.

Configurations

Components in eXo Collaboration Configuration

This table shows some main components that take init-param in the applications of eXo Collaboration:

Applications	Components	Description
Calendar	`CalendarServiceImpl`	It is a service that manages calendars in the Calendar application of eXo Collaboration.

Chat	`HistoryImpl`
	
	`XMPPMessenger`
	
	`DefaultPresenceStatus`
		It is a service that saves the chat history of users.
		
		It is a service that processes messages of chat users, basing on the XMPP Protocol.
		
		It is a component that controls the presence status of chat users.

Contact	`ContactServiceImpl`	It is a service that supplies functions to manage contacts in the Address Book application of eXo Collaboration.
Webservice	`AddActionsPlugin`	It is used to register a listener for the actions on the nodes.

CalendarService

The configuration of the Calendar application is applied mainly in */eXoApplication/calendar/service/src/main/resources/conf/portal/configuration.xml*.

Use the CalendarService to configure the Calendar. The following information will explain details of its configuration. When this configuration file is executed, the component named *org.exoplatform.calendar.service.impl.CalendarServiceImpl* will process actions of the Calendar application.

```
<component>
 <key>org.exoplatform.calendar.service.CalendarService</key>
 <type>org.exoplatform.calendar.service.impl.CalendarServiceImpl</type>
 <init-params>
  <properties-param>
   <name>eventNumber.info</name>
   <property name="eventNumber" value="100"/>
  </properties-param>
 </init-params>
</component>
```

Details:

Properties-Param	Property name	Description	Possible Value	Default Value
eventNumber	`eventNumber`	The number of events in a calendar.	`interger`	`100`

HistoryImpl

The configuration of historyImpl is found in the */extension/webapp/src/main/webapp/WEB-INF/cs-extension/cs/chat/chat-service-configuration.xml*. When this configuration file is executed, the component named *org.exoplatform.services.xmpp.history.impl.jcr.HistoryImpl* initializes all the configured parameters.

```
<component>
 <type>org.exoplatform.services.xmpp.history.impl.jcr.HistoryImpl</type>
 <init-params>
  <value-param>
   <name>workspace</name>
   <value>collaboration</value>
  </value-param>
  <value-param>
   <name>repository</name>
   <value>repository</value>
  </value-param>
  <value-param>
   <name>path</name>
   <value>exo:applications/eXoChat/history</value>
  </value-param>
 </init-params>
```

```
</component>
```

Details:

Value-Param	Description	Possible Values	Default Value
workspace	The workspace name in JCR where history data is stored.	`string`	`collaboration`
repository	The repository name in JCR where history data is stored.	`string`	`repository`
path	The JCR path to the location where history data is stored.	`string`	`exo:applications/eXoChat/history`.

XMPPMessenger

The configuration of the XMPPMessenger component is found in */extension/webapp/src/main/webapp/WEB-INF/cs-extension/cs/chat/chat-service-configuration.xml*. It helps eXo Collaboration connect the Openfire instance.

```
<component>
  <type>org.exoplatform.services.xmpp.connection.impl.XMPPMessenger</type>
  <init-params>
    <properties-param>
      <name>openfire-connection-conf</name>
      <property name="host" value="127.0.0.1"/>
      <property name="port" value="5222"/>
    </properties-param>
    <properties-param>
      <name>send-file</name>
      <property name="timeout" value="7200000"/>
    </properties-param>
  </init-params>
</component>
```

Details:

Properties-param	Property name	Description	Possible Values	Default Value
	`host`		`integer`	127.0.0.1

Properties-param	Property name	Description	Possible Values	Default Value
openfire-connection-conf	`port`	IP address or hostname for the openfire server. Port to connect on the Openfire server. Should be the same that is set in the Openfire configuration "Client to Server".	`integer`	5222
send-file	`timeout`	The timeout before aborting attempt to establish a file transfer.	`integer`	`7200000`

DefaultPresenceStatus

The configuration of the DefaultPresenceStatus component is found in */extension/webapp/src/main/webapp/WEB-INF/cs-extension/cs/chat/chat-service-configuration.xml*.

```
<component>
  <type>org.exoplatform.services.presence.DefaultPresenceStatus</type>
  <init-params>
    <properties-param>
      <name>presence-status</name>
      <property name="mode" value="Free to chat"/>
    </properties-param>
  </init-params>
</component>
```

Details:

Properties-param	Property name	Description	Possible Values	Default Value
presence-status	`mode`		`string`	`Free to chat`

Properties-param	Property name	Description	Possible Values	Default Value
		Show the present status of users.		

ContactService

The configuration of the ContactService component is found in *eXoApplication/contact/service/src/main/resources/conf/portal/configuration.xml*.

When the server starts, the configuration file which contains the declaration of ContactService component is executed. A ContactService component is then created with params and plugins in the configuration file.

```
<component>
  <key>org.exoplatform.contact.service.ContactService</key>
  <type>org.exoplatform.contact.service.impl.ContactServiceImpl</type>
  <init-params>
    <values-param>
      <name>UserCanSeeAllGroupAddressBooks</name>
     <description>User can see all GroupAddressBooks or only GroupAddressBooks that the user
 has at least one membership</description>
      <value>false</value>
    </values-param>
    <values-param>
      <name>NonPublicGroups</name>
       <description>Groups that should not be displayed in broadcast list. Wildcards may be used
 in groups name</description>
    </values-param>
  </init-params>
</component>
```

Values-param	Description	Possible Values	Default Value
UserCanSeeAllGroupAddressBooks	User can see all GroupAddressBooks or only GroupAddressBooks that the user has at least one membership.	`true/false`	`false`
NonPublicGroups	Groups that should not be displayed in	`true/false`	N/A

Values-param	Description	Possible Values	Default Value
	the broadcast list. Wildcards may be used in groups name.		

External Component Plugins

The following table describes the main functions of external component plugins:

Applications	Components	Description
Calendar	`NewUserListener`	Create default personal calendars.
	`NewGroupListener`	Create default group calendars.
	`NewMembershipListener`	Share calendars to members of a specific group.
	`ReminderPeriodJob`	Execute sending reminder emails to users.
	`PopupReminderPeriodJob`	Open a pop-up reminder on the browser of users.
	`AddActionsPlugin`	Enable the systems to automatically update the updated date of events/tasks in a calendar when contents of these events/tasks are changed.
Chat	`HistoryPeriodJob`	Save the chat history of users.
	`RequestFilterComponentPlugin`	Delete the session of a user when the browser is suddenly closed or the session is changed.
	`AuthenticationLoginListener`	Start the session and log in the chat server.
	`AuthenticationLogoutListener`	End the session and log out the chat server.
Contact	`NewUserListener`	Create personal contact data for users.
	`NewMembershipListener`	Create address book for a specific group.
	`UpdateUserProfileListener`	

Applications	Components	Description
		Update the personal profile of a user when it is changed on the portal.
Content	`RSSContentPlugin`	The formatter used to analyze the data from a RSS resource.
	`DescriptionPlugin`	Represent the data from a RSS resource.
Mail	`AuthenticationLogoutListener`	Stop checking mails of a user when he logs out.
Social Intergration	`CalendarDataInitialize`	Create a calendar for a group in a specific space.
	`ContactDataInitialize`	Create an address book for a group in a specific space.
	`ContactSpaceActivityPublisher`	Customize the activity status of a specific space when an event happens on an address book.
	`CalendarSpaceActivityPublisher`	Customize the activity status of a specific space when an event happens on a calendar.
	`PortletPreferenceRequiredPlugin`	Declare the application that will automatically create database.

Calendar Configuration

NewUserListener

Each user can have a default personal calendar created. Use the NewUserListener to configure that. To use the plugin in the component configuration, you must use the target-component:

```
<target-component>org.exoplatform.services.organization.OrganizationService</target-component>
```

The configuration is applied mainly in *extension/webapp/src/main/webapp/WEB-INF/cs-extension/cs/calendar/calendar-service-configuration.xml*.

```
<component-plugin>
  <name>calendar.new.user.event.listener</name>
  <set-method>addListenerPlugin</set-method>
  <type>org.exoplatform.calendar.service.impl.NewUserListener</type>
  <description>description</description>
  <init-params>
    <value-param>
      <name>defaultEventCategories</name>
      <value>Meeting,Calls,Clients,Holiday,Anniversary</value>
    </value-param>
    <value-param>
      <name>defaultCalendarCategory</name>
      <value>My group</value><!-- Single value -->
    </value-param>
    <value-param>
      <name>defaultCalendar</name>
      <value>Default</value>
    </value-param>
    <!--Params for default calendar setting -->
    <value-param>
      <name>viewType</name>
      <value>1</value>
    </value-param>

    <value-param>
      <name>timeInterval</name>
      <value>15</value><!-- in minutes -->
    </value-param>

    <value-param>
      <name>weekStartOn</name>
      <value>2</value>
    </value-param>

    <value-param>
      <name>dateFormat</name>
      <value>MM/dd/yyyy</value>
    </value-param>

    <value-param>
      <name>timeFormat</name>
      <value>HH:mm</value> <!-- HH:mm/hh:mm a -->
    </value-param>
```

```
    <value-param>
      <name>localeId</name>
      <value>BEL</value>
    </value-param>

    <value-param>
      <name>timezoneId</name>
      <value>Europe/Brussels</value>
    </value-param>

    <value-param>
      <name>baseUrlForRss</name>
      <value/>
    </value-param>

    <value-param>
      <name>isShowWorkingTime</name>
      <value>false</value><!-- boolean true/false -->
    </value-param>

    <value-param>
      <name>workingTimeBegin</name>
      <value>08:00</value><!-- -->
    </value-param>

    <value-param>
      <name>workingTimeEnd</name>
      <value>18:00</value><!-- -->
    </value-param>

    <values-param>
      <name>ignoredUsers</name>
      <description>Definition users to ignore create default calendar</description>
      <!-- <value>demo</value> <value>marry</value> -->
    </values-param>
  </init-params>
</component-plugin>
```

Details:

- **Name**: `calendar.new.user.event.listener` - The unique key to avoid duplicate names. Users can change it.

- **Set-method**: `addListenerPlugin` - The function is executed at the target of the component to register NewUserListener.
- **Type**: `org.exoplatform.calendar.service.impl.NewUserListener` - The class is set up to execute the creation of database.
- **Description**: It is a plugin used to create default personal calendars.

See the details about the init-params of the component in the following table:

Value-params	Possible value	Default value	Description
defaultEventCategories	`string (Comma separated list of category names)`	`Meeting,Calls,Clients,Holiday,Anniversary`	Default event categories for users.
defaultCalendarCategory	`string`	`Default`	Name of the calendar group.
viewType	`0-6 (see below)`	`1`	Default view after user logs in and goes to the Calendar portlet.
timeInterval	`integer in minutes`	`15`	The time unit interval when you drag and move the event (in Day and Week views only).
weekStartOn	`1-7 (see below)`	`2`	Day to use as the beginning of the week. It only affects the Week view.
dateFormat	`valid Java Date format`	`MM/dd/yyyy`	The display format for dates.
timeFormat	`valid Java Date format`	`HH:mm`	The display format for time.
localeId	`valid locale ID`	`BEL`	Id of the geographic locale.
timezoneIds	`valid TimeZone id`	`Europe`	User time zone.
baseUrlForRss	`none`	`none`	The URL to publish the RSS content.
isShowWorkingTime	`true/false`	`false`	Indicate if the working time should be highlighted in the Day view.
workingTimeBegin	`time in timeFormat`	`08:00`	The start time in working time.

Value-params	Possible value	Default value	Description
workingTimeEnd	`time in timeFormat`	`18:00`	The end time in working time.
ignoredUsers	`user id, use multiple by each line`	`demo/marry`	Definition users to ignore creating the default calendar.

The **viewType** parameter is encoded by a number as follow:

0 : Day view

1 : Week view

2 : Month view

3 : Year view

4 : List view

5 : Schedule view

6 : Working days view

The **weekStartOn** parameter is encoded as follow:

1 : Sunday

2 : Monday

3 : Tuesday

4 : Wednesday

5 : Thursday

6 : Friday

7 : Saturday

NewGroupListener

To use the plugin in the component configuration, you must use the target-component:

```
<target-component>org.exoplatform.services.organization.OrganizationService</target-component>
```

The configuration is applied mainly in *extension/webapp/src/main/webapp/WEB-INF/cs-extension/cs/calendar/calendar-service-configuration.xml*.

```
<component-plugin>
  <name>calendar.new.group.event.listener</name>
  <set-method>addListenerPlugin</set-method>
  <type>org.exoplatform.calendar.service.impl.NewGroupListener</type>
  <description>description</description>
  <init-params>
    <value-param>
      <name>defaultEditPermission</name>
      <value>*.*</value><!-- Multi value membership, use coma (,) to split values -->
    </value-param>
    <value-param>
      <name>defaultViewPermission</name>
      <value>*.*</value>
    </value-param>
    <value-param>
      <name>defaultLocale</name>
      <value>BEL</value>
    </value-param>
    <value-param>
      <name>defaultTimeZone</name>
      <value>Europe/Brussels</value>
    </value-param>
    <values-param>
      <name>ignoredGroups</name>
      <description>Definition group to ignore create default calendar</description>
      <!-- <value>/platform/guests</value> -->
      <value>/spaces/*</value> <!-- single value, use more <value> tags to add more group -->
    </values-param>
  </init-params>
</component-plugin>
```

Details:

- **Name**: `calendar.new.group.event.listener` - The unique key to avoid duplicate names. Users can change it.
- **Set-method**: `addListenerPlugin` - The function is executed at the target of the component to register NewGroupListener.
- **Type**: `org.exoplatform.calendar.service.impl.NewGroupListener` - The class which is set up to execute the creation of database.
- **Description** - It is the plugin used to create default group calendars.

See the details about the init-params of the component in the following table:

Value-params	Possible values	Default values	Description
defaultEditPermission	User id (Multi value membership, use coma (,) to split values)	. means that all members in that group can modify and add, remove a calendar, events/ tasks of the calendar	The default permission assigned to membership in a specific group to edit calendars and events/ tasks of the calendar.
defaultViewPermission	User id (Multi value membership, use coma (,) to split values)	. means that all members in that group can view this calendar and all the events/tasks of this calendar.	The default permission assigned to membership in a specific group to view a calendar and events /tasks of the calendar.
defaultLocale	Valid locade id	BEL (see more locale ids *http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html*)	The default locale of the calendar.
defaultTimeZone	Valid timezone id	Europe/Brussels (see more for timeZone ids *http://www.unicode.org/cldr/data/docs/design/formatting/zone_log.html#windows_ids)*	The default time zone of the calendar.
ignoredGroups	Group id (use line to define multiple value)	/platform/guests	Definition group to ignore create the default calendar.

NewMembershipListener

To use the plugin in the component configuration, you must use the target-component:

```
<target-component>org.exoplatform.services.organization.OrganizationService</target-component>
```

The configuration is applied mainly in *extension/webapp/src/main/webapp/WEB-INF/cs-extension/cs/calendar/calendar-service-configuration.xml*.

```
<component-plugin>
  <name>calendar.new.membership.event.listener</name>
  <set-method>addListenerPlugin</set-method>
  <type>org.exoplatform.calendar.service.impl.NewMembershipListener</type>
  <description>description</description>
</component-plugin>
```

Details:

- **Name**: `calendar.new.membership.event.listener` - The unique key to avoid duplicate names. Users can change it.
- **Set-method**: `addListenerPlugin` - The function which is executed at the target of the component.
- **Type**: `org.exoplatform.calendar.service.impl.NewMembershipListener` - The class which is set up to execute the creation of database.
- **Description**: It is a plugin used to execute sending reminder emails to users.

ReminderPeriodJob

The Calendar application of eXo Collaboration can send event reminders by using the email reminder plugin configuration. You will probably need to adjust this configuration to meet your own needs. The feature is based on a periodic poll of the stored reminders.

You must use the following target component to use the plugin in this configuration:

```
<target-component>org.exoplatform.services.scheduler.JobSchedulerService</target-component>
```

The configuration is applied in *extension/webapp/src/main/webapp/WEB-INF/cs-extension/cs/cs-configuration.xml*.

```
<component-plugin>
  <name>RecordsJob</name>
  <set-method>addPeriodJob</set-method>
```

```
  <type>org.exoplatform.calendar.service.ReminderPeriodJob</type>
  <description>add e-mail reminder job to the JobSchedulerService</description>
  <init-params>
    <properties-param>
      <name>job.info</name>
      <description>save the monitor data periodically</description>
      <property name="jobName" value="ReminderJob"/>
      <property name="groupName" value="CollaborationSuite"/>
      <property name="job" value="org.exoplatform.calendar.service.ReminderJob"/>
      <property name="repeatCount" value="0"/>
      <property name="period" value="180000"/>
      <property name="startTime" value="+0"/>
      <property name="endTime" value=""/>
    </properties-param>
  </init-params>
</component-plugin>
```

Details:

- **Name**: `RecordsJob` - The name of a schedule job.
- **Set-method**: `addPeriodJob` - The plugin which registers the method.
- **Type**: `org.exoplatform.calendar.service.ReminderPeriodJob` - A class that executes to transfer data into database of Job Scheduler.
- **Description**: Add email reminder job to the JobSchedulerService.

See details about the init-params of the component in the following table:

Properties-param	Description	Property names	Description	Possible values	Default values
job.info	Save the monitor data periodically.	`jobName`	The name of job	`String`	`ReminderJob`
		`groupName`	The name of group job.	`String`	`CollaborationSuite`
		`job`	The name of actual job class.	`Class path`	`org.exoplatform.calendar.ser`
		`repeatCount`	How many times to run this job.	`Long`	`0, ( use '0' which means 'run forever)`

Properties-param	Description	Property names	Description	Possible values	Default values
		`period`	The time interval (millisecond) between job executions.	`Long`	`180000`
		`startTime`	The time when the job starts running.	`Integer`	`+0`
		`endTime`	The time when the job ends running.	`Integer`	`none`

PopupReminderPeriodJob

You must use the following target component to use the plugin in this configuration:

```
<target-component>org.exoplatform.services.scheduler.JobSchedulerService</target-component>
```

The configuration is applied in *extension/webapp/src/main/webapp/WEB-INF/cs-extension/cs/cs-configuration.xml*.

```
<component-plugin>
  <name>PopupRecordsJob</name>
  <set-method>addPeriodJob</set-method>
  <type>org.exoplatform.calendar.service.PopupReminderPeriodJob</type>
  <description>add popup reminder job to the JobSchedulerService</description>
  <init-params>
    <properties-param>
      <name>job.info</name>
      <description>save the monitor data periodically</description>
      <property name="jobName" value="PopupReminderJob"/>
      <property name="groupName" value="CollaborationSuite"/>
      <property name="job" value="org.exoplatform.calendar.service.PopupReminderJob"/>
      <property name="repeatCount" value="0"/>
      <property name="period" value="6000"/>
      <property name="startTime" value="+0"/>
      <property name="endTime" value=""/>
```

```
    </properties-param>
    <properties-param>
      <name>popupreminder.info</name>
      <description>save the monitor data periodically</description>
      <property name="portalName" value="portal"/>
    </properties-param>
  </init-params>
</component-plugin>
```

Details:

- **Name**: `PopupRecordsJob` - The name of the job.
- **Set-method**: `addPeriodJob` - The plugin registering method.
- **Type**: `org.exoplatform.calendar.service.PopupReminderPeriodJob` - The class which executes to transfer the data into database of Job Scheduler.
- **Description**: Add popup reminder job to the JobSchedulerService.

See details about the init-params of the component in the following table:

- Properties-param: **job.info**. This param saves the monitor data periodically and includes the following sub-params:

Properties-param	Description	Property names	Description	Possible values	Default values
job.info	Save the monitor data periodically.	`jobName`	The name of job.	`String`	`PopupReminderJob`
		`groupName`	The name of group job.	`String`	`CollaborationSuite`
		`job`	The name of actual job class.	`Class path`	`org.exoplatform.calendar.service.Popup`
		`repeatCount`	How many times to run this job.	`Long`	`0, ( use '0' which means 'run forever)`
		`period`	The time interval (millisecond)	`Long`	`6000`

Properties-param	Description	Property names	Description	Possible values	Default values	
			between job executions.			
		`startTime`	The time when the job starts running.	`Long`	`+0`	
		`endTime`	The time when the job ends running.	`Integer`	`None`	
popupremindertime	Save the monitor data periodically.	`portalName`	The name of the portal.	`String`	`portal`	##

AddActionsPlugin

The configuration of the AddActionsPlugin is found in *WEB-INF/cs-extension/cs/webservice/webservice-configuration.xml*.

It is used to register the listener named `org.exoplatform.webservice.cs.LastUpdateAction` and it is executed, basing on eventTypes.

```
<component>
  <type>org.exoplatform.services.jcr.impl.ext.action.SessionActionCatalog</type>
  <component-plugins>
    <component-plugin>
      <name>Last Update Action</name>
      <set-method>addPlugin</set-method>
      <type>org.exoplatform.services.jcr.impl.ext.action.AddActionsPlugin</type>
      <description>add actions plugin</description>
      <init-params>
        <object-param>
          <name>actions</name>
                                                                                <object
 type="org.exoplatform.services.jcr.impl.ext.action.AddActionsPlugin$ActionsConfig">
            <field name="actions">
              <collection type="java.util.ArrayList">
                <value>
                  <object type="org.exoplatform.services.jcr.impl.ext.action.ActionConfiguration">
```

```
            <field name="eventTypes">
              <string>addNode,changeProperty</string>
            </field>
            <field name="nodeTypes">
              <string>exo:calendarEvent</string>
            </field>
            <field name="actionClassName">
              <string>org.exoplatform.webservice.cs.LastUpdateAction</string>
            </field>
          </object>
        </value>
      </collection>
    </field>
  </object>
 </object-param>
</init-params>
</component-plugin>
</component-plugins>
</component>
```

- **object-param**: `Actions` - The name of the object.
- **object type**: `org.exoplatform.services.jcr.impl.ext.action.AddActionsPlugin$ActionsConfig` - A class used to register the following field names in the table below:

Field name	String	Description
eventTypes	`addNode,changeProperty`	The type of the event.
nodeTypes	`exo:calendarEvent`	The type of the node.
actionClassName	`org.exoplatform.webservice.cs.LastUpdateAction`	The registration class to execute the actions that the plugin requires.

Chat Configuration

The Chat congiguration is applied in */extension/webapp/src/main/webapp/WEB-INF/cs-extension/cs/chat/chat-service-configuration.xml*.

HistoryPeriodJob

```
<external-component-plugins>
```

```
            <target-component>org.exoplatform.services.scheduler.JobSchedulerService</target-
component>
  <component-plugin>
    <name>ChatRecordsJob</name>
    <set-method>addPeriodJob</set-method>
    <type>org.exoplatform.services.xmpp.connection.impl.HistoryPeriodJob</type>
    <description>add chat messages from Openfire Server to History</description>
    <init-params>
      <properties-param>
        <name>job.info</name>
        <description>save the monitor data periodically</description>
        <property name="jobName" value="messageToHistoricalMessageJob"/>
        <property name="groupName" value="CollaborationSuite"/>
        <property name="job" value="org.exoplatform.services.xmpp.connection.impl.HistoryJob"/>
        <property name="repeatCount" value="0"/>
        <property name="period" value="3000"/>
        <property name="startTime" value="+0"/>
        <property name="endTime" value=""/>
      </properties-param>
      <properties-param>
        <name>history.info</name>
        <description>save the monitor data periodically</description>
        <property name="logBatchSize" value="50"/>
      </properties-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

Details:

- **Name**: `ChatRecordsJob` - The name of the job.
- **Set-method**: `addPeriodJob` - The plugin which registers the method.
- **Type**: `org.exoplatform.services.xmpp.connection.impl.HistoryPeriodJob` - A class which executes to transfer the data into the database of Job Scheduler.
- **Description**: It is used to save chat messages from Openfire Server to History.

See details about the init-params of the component in the following table:

Properties-param	Description	Property names	Description	Possible values	Default values
job.info	Save the monitor data periodically.	`jobName`	The name of job.	`String`	`messageToHistoricalMessageJo`

Properties-param	Description	Property names	Description	Possible values	Default values
		`groupName`	The name of group name.	`String`	`CollaborationSuite`
		`job`	The name of actual job class.	`Class path`	`org.exoplatform.services.xmp`
		`repeatCount`	How many times to run this job.	`integer`	`0` (use '0' which means 'run forever)
		`period`	The time interval (millisecond) between job executions.	`Long`	`3000`
		`startTime`	The time the job starts running.	`Long`	`+0`
		`endTime`	The time the job ends running.	`Long`	`none`
history.info	Save the monitor data periodically.	`logBatchSize`	The maximum number of messages in the cache are saved once the job runs.	`Integer`	`50`

RequestFilterComponentPlugin

```
<external-component-plugins>
		<target-component>org.exoplatform.services.rest.impl.RequestHandlerImpl</target-component>
  <component-plugin>
    <name>ws.rs.request.filter</name>
    <set-method>addPlugin</set-method>
    <type>org.exoplatform.services.rest.impl.RequestFilterComponentPlugin</type>
    <init-params>
      <value-param>
        <name>RESTXMPPServiceFilter</name>
        <value>org.exoplatform.services.xmpp.rest.filter.RESTXMPPServiceFilter</value>
```

```
    </value-param>
   </init-params>
  </component-plugin>
</external-component-plugins>
```

Details:

- **Name**: `ws.rs.request.filter` - The name of the filter.
- **Set-method**: `addPlugin` - The plugin which registers the method.
- **Type**: `org.exoplatform.services.rest.impl.RequestFilterComponentPlugin` - The class which executes the requests of the plugin.
- **Description**: It is used to delete the session of a user when he suddenly closes the browser or changes the session.

See details about the init-params of the component in the following table:

Value-param	Description	Possible value	Default value
RESTXMPPServiceFilter	The name of the filter.	`Class path`	`org.exoplatform.services.xmpp.rest.`

AuthenticationLoginListener and AuthenticationLogoutListener

Two functions, including login and logout of XMPPRestService, are responsible for creating a new XMPPSessionImpl and destroying an existing XMPPSessionImpl. They can be called by listeners: AuthenticationLoginListener, AuthenticationLogoutListener or from client(browser) through the REST protocol (jabberLogin, jabberLogout in UIMainChatWindow.js). You must use the same target component for two external component plugins:

```
<target-component>org.exoplatform.services.listener.ListenerService</target-component>
```

AuthenticationLoginListener

```
<component-plugin>
  <name>exo.core.security.ConversationRegistry.register</name>
  <set-method>addListener</set-method>
  <type>org.exoplatform.services.xmpp.connection.impl.AuthenticationLoginListener</type>
  <description>description</description>
</component-plugin>
```

Details:

- **Name**: `exo.core.security.ConversationRegistry.register` - The name of plugin.
- **Set-method**: `addListener` - The plugin which registers the method.
- **Type**: `org.exoplatform.services.xmpp.connection.impl.AuthenticationLoginListener` - The class to execute the requests of the plugin.
- **Description**: It is used to start the session and log in the chat server.

AuthenticationLogoutListener

```
<component-plugin>
  <name>exo.core.security.ConversationRegistry.unregister</name>
  <set-method>addListener</set-method>
  <type>org.exoplatform.services.xmpp.connection.impl.AuthenticationLogoutListener</type>
  <description>description</description>
</component-plugin>
```

Details:

- **Name**: `exo.core.security.ConversationRegistry.register` - The name of plugin.
- **Set-method**: `addListener` - The plugin which registers the method.
- **Type**: `org.exoplatform.services.xmpp.connection.impl.AuthenticationLogoutListener` - A class to execute the requests of the plugin.
- **Description**: It is used to end the session and log in the chat server.

Contact Configuration

The Contact Application is configured by three external component plugins: `NewUserListener`, `NewMembershipListener` and `UpdateUserProfileListener`. They use the same target component:

```
<target-component>org.exoplatform.services.organization.OrganizationService</target-component>
```

The Contact configuration is found in *extension/webapp/src/main/webapp/WEB-INF/cs-extension/cs/contact/contact-service-configuration.xml*.

NewUserListener

```
<component-plugin>
  <name>contact.new.user.event.listener</name>
  <set-method>addListenerPlugin</set-method>
  <type>org.exoplatform.contact.service.impl.NewUserListener</type>
  <description>description</description>
</component-plugin>
```

Details:

- **Name**: `contact.new.user.event.listener` - The name of listener.
- **Set-method**: `addListenerPlugin` - The plugin which registers the method.
- **Type**: `org.exoplatform.contact.service.impl.NewUserListener` - A class that executes all requirements of the plugin.
- **Description**: It is used to initialize personal contact data for users.

NewMembershipListener

```
<component-plugin>
  <name>contact.new.membership.event.listener</name>
  <set-method>addListenerPlugin</set-method>
  <type>org.exoplatform.contact.service.impl.NewMembershipListener</type>
  <description>description</description>
</component-plugin>
```

Details:

- **Name**: `contact.new.membership.event.listener` - The name of the listener.
- **Set-method**: `addListenerPlugin` - The plugin which registers the method. Keep this value.
- **Type**: `org.exoplatform.contact.service.impl.NewMembershipListener` - The class which executes all requirements of the plugin.
- **Description**: It is used to initialize an address book for a specific group.

UpdateUserProfileListener

```
<component-plugin>
  <name>contact.new.userprofile.event.listener</name>
  <set-method>addListenerPlugin</set-method>
  <type>org.exoplatform.contact.service.impl.UpdateUserProfileListener</type>
  <description>description</description>
</component-plugin>
```

Details:

- **Name**: `contact.new.userprofile.event.listener` - The name of the listener.
- **Set-method**: `addListenerPlugin` - The plugin which registers the method. Keep this value.
- **Type**: `org.exoplatform.contact.service.impl.UpdateUserProfileListener` - The class which executes all the requirements of the plugin.
- **Description**: It is used to update the personal profile of a user when he changes it on the portal.

Content Configuration

The Content application, such as RSS reader of eXo Collaboration, is configured by two external component plugins: RSSContentPlugin and DescriptionPlugin. Both the external components plugins use the same target component:

```
<target-component>org.exoplatform.content.service.ContentDAO</target-component>
```

This content configuration is applied in *extension/webapp/src/main/webapp/WEB-INF/cs-extension/cs/content/content-service-configuration.xml*.

RSSContentPluginDescriptionPlugin

```
<component-plugin>
  <name>rssreader.listener</name>
  <set-method>addPlugin</set-method>
  <type>org.exoplatform.content.service.RSSContentPlugin</type>
  <description>rss reader plugin</description>
</component-plugin>
```

Details:

- **Name**: `rssreader.listener` - The name of the listener.
- **Set-method**: `addPlugin` - The plugin which registers the method. Keep this value.
- **Type**: `org.exoplatform.content.service.RSSContentPlugin` - A class which extends ContentPlugin and implements the loadContentMeta method to get content items.
- **Description**: It is a formater used to analyze the data from a RSS resource.

DescriptionPlugin

```
<component-plugin>
  <name>description.listener</name>
  <set-method>addPlugin</set-method>
  <type>org.exoplatform.content.service.DescriptionPlugin</type>
  <description>Description plugin</description>
</component-plugin>
```

Details:

- **Name**: `description.listener` - The name of the listener.
- **Set-method**: `addPlugin` - The plugin registering method. Keep this value.
- **Type**: `org.exoplatform.content.service.DescriptionPlugin` - A class which executes all requirements of the plugin.
- **Description**: It is a plugin to represent data from a RSS source.

Mail Configuration

AuthenticationLogoutListener

In the Mail application of eXo Collaboration, when a user checks messages for one account, the remote mailbox fetch is performed as a background job. Before eXo Collaboration 1.2, the job was continued until all messages had been retrieved or when the user stopped the check through the UI. Hence, even when a user was not logged in, the background job was continued. This can be the resource intensive for the server if many users have large mailboxes. Since eXo Collaboration 1.2, one capability is added to halt the background job when the user session terminates (logout or timeout). It makes eXo Collaboration more friendly with server resources. If you want to activate this feature, you need to add a bunch of the xml congifuration in */extension/webapp/src/main/webapp/WEB-INF/cs-extension/cs/mail/mail-service-configuration.xml*:

```
<external-component-plugins>
  <target-component>org.exoplatform.services.listener.ListenerService</target-component>
   <component-plugin>
    <name>exo.core.security.ConversationRegistry.unregister</name>
    <set-method>addListener</set-method>
    <type>org.exoplatform.mail.service.AuthenticationLogoutListener</type>
    <description>description</description>
  </component-plugin>
</external-component-plugins>
```

Details:

- **Name**: `exo.core.security.ConversationRegistry.unregister` - The name of listener.
- **Set-method**: `addListener` The plugin which registers the method. Keep this value.
- **Type**: `org.exoplatform.mail.service.AuthenticationLogoutListener` - A class which executes all requirements of the plugin.
- **Description**: It is a plugin used to stop checking mails of a user when he logs out.

MailSettingConfigPlugin

Since eXo Collaboration 2.2.0, *MailSettingConfigPlugin* is used to define the behavior, for example, showing/hiding fields and checking/unchecking checkboxes, of email account settings in the mail-server-configuration.xml file. It allows administrators to preconfigure all settings and to specify if end-users have the modification right on each specific setting or not.

```
<external-component-plugins>
  <target-component>org.exoplatform.mail.service.MailService</target-component>
 <component-plugin>
  <name>cs.mail.service.settings</name>
     <set-method>addPlugin</set-method>
  <type>org.exoplatform.mail.service.MailSettingConfigPlugin</type>
  <description>description</description>
   <init-params>
    <object-param>
     <name>leaveOnServer</name>
       <description>options to keep a copy of the message on the mail server after eXo Mail has
 downloaded the message</description>
      <object type="org.exoplatform.mail.service.MailSettingConfig">
        <field name="name"><string>leaveOnServer</string></field>
```

```
      <field name="userAllowed"><boolean>true</boolean></field>
      <field name="defaultValue"><string>true</string></field>
    </object>
  </object-param>
  <object-param>
   <name>incomingServer</name>
    <description>default incoming server to check for new mails.</description>
    <object type="org.exoplatform.mail.service.MailSettingConfig">
      <field name="name"><string>incomingServer</string></field>
      <field name="userAllowed"><boolean>true</boolean></field>
      <field name="defaultValue"><string>imap.gmail.com</string></field>
    </object>
  </object-param>
  <object-param>
   <name>incomingPort</name>
    <description>default port incoming server to check for new mails.</description>
    <object type="org.exoplatform.mail.service.MailSettingConfig">
      <field name="name"><string>incomingPort</string></field>
      <field name="userAllowed"><boolean>true</boolean></field>
      <field name="defaultValue"><string>993</string></field>
    </object>
  </object-param>
  <object-param>
   <name>outgoingServer</name>
    <description>description</description>
    <object type="org.exoplatform.mail.service.MailSettingConfig">
      <field name="name"><string>outgoingServer</string></field>
      <field name="userAllowed"><boolean>true</boolean></field>
      <field name="defaultValue"><string>smtp.gmail.com</string></field>
    </object>
  </object-param>
  <object-param>
   <name>outgoingPort</name>
    <description>description</description>
    <object type="org.exoplatform.mail.service.MailSettingConfig">
      <field name="name"><string>outgoingPort</string></field>
      <field name="userAllowed"><boolean>true</boolean></field>
      <field name="defaultValue"><string>465</string></field>
    </object>
  </object-param>
  <object-param>
   <name>acceptIncomingSecureAuthentication</name>
    <description>description</description>
    <object type="org.exoplatform.mail.service.MailSettingConfig">
```

```
      <field name="name"><string>acceptIncomingSecureAuthentication</string></field>
      <field name="userAllowed"><boolean>true</boolean></field>
      <field name="defaultValue"><string>true</string></field>
    </object>
  </object-param>
  <object-param>
   <name>incomingSecureAuthentication</name>
    <description>description</description>
    <object type="org.exoplatform.mail.service.MailSettingConfig">
      <field name="name"><string>incomingSecureAuthentication</string></field>
      <field name="userAllowed"><boolean>true</boolean></field>
      <field name="defaultValue"><string>starttls</string></field>
    </object>
  </object-param>
  <object-param>
   <name>incomingAuthenticationMechanism</name>
    <description>description</description>
    <object type="org.exoplatform.mail.service.MailSettingConfig">
      <field name="name"><string>incomingAuthenticationMechanism</string></field>
      <field name="userAllowed"><boolean>true</boolean></field>
      <field name="defaultValue"><string>plain</string></field>
    </object>
  </object-param>
  <object-param>
   <name>acceptOutgoingSecureAuthentication</name>
    <description>description</description>
    <object type="org.exoplatform.mail.service.MailSettingConfig">
      <field name="name"><string>acceptOutgoingSecureAuthentication</string></field>
      <field name="userAllowed"><boolean>true</boolean></field>
      <field name="defaultValue"><string>true</string></field>
    </object>
  </object-param>
  <object-param>
   <name>outgoingSecureAuthentication</name>
    <description>description</description>
    <object type="org.exoplatform.mail.service.MailSettingConfig">
      <field name="name"><string>outgoingSecureAuthentication</string></field>
      <field name="userAllowed"><boolean>true</boolean></field>
      <field name="defaultValue"><string>starttls</string></field>
    </object>
  </object-param>
  <object-param>
   <name>outgoingAuthenticationMechanism</name>
    <description>description</description>
```

```
      <object type="org.exoplatform.mail.service.MailSettingConfig">
        <field name="name"><string>outgoingAuthenticationMechanism</string></field>
        <field name="userAllowed"><boolean>true</boolean></field>
        <field name="defaultValue"><string>plain</string></field>
      </object>
    </object-param>
  </init-params>
  </component-plugin>
 </external-component-plugins>
```

Details:

- **Name**: `cs.mail.service.settings` - The name of the mail settings.
- **Set-method**: `addPlugin` - The plugin which registers the method.
- **Type**: `org.exoplatform.mail.service.MailSettingConfigPlugin` - The plugin's class name.
- **object type**: `org.exoplatform.mail.service.MailSettingConfig` - A class of object that contains a specific setting.
- **Description**: Describes what the setting is or what the setting does.

Object-param	Desciption
leaveOnServer	Options to keep the message on the mail server after it has been downloaded to the Mail application of eXo Collaboration.
incomingServer	The default incoming server used to check new mails.
incomingPort	The default port of the incoming server used to check new mails.
outgoingServer	The default port of the outgoing server used to send new mails.
outgoingPort	The default outgoing port to send new mails.
acceptIncomingSecureAuthentication	Accept the secure authentication of the incoming server.
incomingSecureAuthentication	The type of incoming secure authentication.
incomingAuthenticationMechanism	The type of incoming authentication mechanism.
acceptOutgoingSecureAuthentication	Accepts the secure authentication of the outgoing server.

Object-param	Desciption
outgoingSecureAuthentication	The type of outgoing secure authentication.
outgoingAuthenticationMechanism	The type of incoming authentication mechanism.

The object parameters have the same field names, but the values of the parameters are different.

Field names	Description	Possible values
name	The field name in the account settings form.	String
userAllowed	Allow users to edit the field in the account settings form or not.	Boolean
defaultValue	The default value of the field in the account settings form.	String

Social Integration Configuration

The Social Integration Configuration is applied in */extension/webapp/src/main/webapp/WEB-INF/cs-extension/cs/social-integration/social-integration-configuration.xml*.

CalendarDataInitialize

```
<external-component-plugins>
  <target-component>org.exoplatform.social.core.space.spi.SpaceService</target-component>
  <component-plugin>
    <name>CalendarDataInitialize</name>
    <set-method>addSpaceListener</set-method>
    <type>org.exoplatform.cs.ext.impl.CalendarDataInitialize</type>
    <init-params>
      <value-param>
        <name>portletName</name>
        <value>CalendarPortlet</value>
      </value-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

Details:

- **Name**: `CalendarDataInitialize` - The name of plugin.

- **Set-method**: `addSpaceListener` - The plugin which registers the method.
- **Type**: `org.exoplatform.cs.ext.impl.CalendarDataInitialize` - A class that executes all requirements of the plugin.
- **Description**: It is used to initialize a calendar for a group in a specific space.

See the details about the init-params of the component in the following table:

value-param	Description	Possible value	Default value
portletName	The name of the portlet	`String`	`CalendarPortlet`

ContactDataInitialize

```
<external-component-plugins>
  <target-component>org.exoplatform.social.core.space.spi.SpaceService</target-component>
  <component-plugin>
    <name>ContactDataInitialize</name>
    <set-method>addSpaceListener</set-method>
    <type>org.exoplatform.cs.ext.impl.ContactDataInitialize</type>
    <init-params>
      <value-param>
        <name>portletName</name>
        <value>ContactPortlet</value>
      </value-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

Details:

- **Name**: `ContactDataInitialize` - The name of the plugin.
- **Set-method**: `addSpaceListener` - The plugin which registers the method.
- **Type**: `org.exoplatform.cs.ext.impl.ContactDataInitialize` - A class that executes all the requires of the plugin.
- **Description**: It is a plugin used to initialize an address book for a group in a specific space.

See the details about the init-params of the component in the following table:

value-param	Possible value	Default value	Description
portletName	`String`	`ContactPortlet`	The name of the portlet.

ContactSpaceActivityPublisher

```
<external-component-plugins>
  <target-component>org.exoplatform.contact.service.ContactService</target-component>
  <component-plugin>
    <name>ContactEventListener</name>
    <set-method>addListenerPlugin</set-method>
    <type>org.exoplatform.cs.ext.impl.ContactSpaceActivityPublisher</type>
  </component-plugin>
</external-component-plugins>
```

Details:

- **Name**: `ContactEventListener` - The name of the plugin.
- **Set-method**: `addListenerPlugin` - The plugin which registers the method.
- **Type**: `org.exoplatform.cs.ext.impl.ContactSpaceActivityPublisher` - A class that executes all requirements of the plugin.
- **Description**: It is a plugin used to customize the activity status of a specific space when an event happens on an address book.

CalendarSpaceActivityPublisher

```
<external-component-plugins>
  <target-component>org.exoplatform.calendar.service.CalendarService</target-component>
  <component-plugin>
    <name>CalendarEventListener</name>
    <set-method>addEventListenerPlugin</set-method>
    <type>org.exoplatform.cs.ext.impl.CalendarSpaceActivityPublisher</type>
  </component-plugin>
</external-component-plugins>
```

Details:

- **Name**: `CalendarEventListener` - The name of the plugin.

- **Set-method**: `addEventListenerPlugin` - The plugin which registers the method.
- **Type**: `org.exoplatform.cs.ext.impl.CalendarSpaceActivityPublisher` - A class that executes all the requires of the plugin.
- **Description**: It is a plugin used to customize the activity status of a specific space when an event happens on a calendar.

PortletPreferenceRequiredPlugin

```
<external-component-plugins>
  <target-component>org.exoplatform.social.core.space.spi.SpaceService</target-component>
  <component-plugin>
    <name>portlets.prefs.required</name>
    <set-method>setPortletsPrefsRequired</set-method>
    <type>org.exoplatform.social.core.application.PortletPreferenceRequiredPlugin</type>
    <init-params>
      <values-param>
        <name>portletsPrefsRequired</name>
        <value>CalendarPortlet</value>
        <value>ContactPortlet</value>
      </values-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

Details:

- **Name**: `ortlets.prefs.required` - The name of the plugin.
- **Set-method**: `setPortletsPrefsRequired` - The plugin which registers the method.
- **Type**: `org.exoplatform.social.core.application.PortletPreferenceRequiredPlugin` - A class that executes all the requires of the plugin.
- **Description**: It is a plugin used to declare the application that will automatically create database.

See the details about the init-params of the component in the following table:

value-param	Possible value	Default value	Description
portletsPrefsRequired	`String`	`ContactPortlet` / `ContactPortlet`	The name of plugin added to SpaceService.

Data Injectors

Data injectors are used to create data for performance test. This part will describe which data injectors are implemented in eXo Collaboration and how to use them.

In eXo Collaboration, data injectors are implemented as plug-ins attached to the *org.exoplatform.services.bench.DataInjectorService* service and handled via RESTful requests. This service is normally registered to the portal container as a general component.

To use this service, add the following dependency to the Classpath of the server:

```
<dependency>
    <groupId>org.exoplatform.commons</groupId>
    <artifactId>exo.platform.commons.component</artifactId>
</dependency>
```

To activate the *DataInjectorService* component, you need to register it to a portal container by the following configuration:

```
<component>
   <type>org.exoplatform.services.bench.DataInjectorService</type>
</component>
```

When you want to inject data for a specific product, you must implement a class which extends "org.exoplatform.services.bench.DataInjector" and register it to the *DataInjectorService* component as a plug-in.

In eXo Collaboration, there are three plug-ins attached to the *DataInjectorService* component:

- ContactDataInjector
- CalendarDataInjector
- MailDataInjector

ContactDataInjector

The *ContactDataInjector* plug-in is used to manage data injection of the Address Book application.

To use this plug-in, do as follows:

1. Add the following configuration to the *configuration.xml* file to register the plug-in to the *DataInjectorService* component:

```
<external-component-plugins>
   <target-component>org.exoplatform.services.bench.DataInjectorService</target-component>
   <component-plugin>
    <name>ContactDataInjector</name>
    <set-method>addInjector</set-method>
    <type>org.exoplatform.contact.service.bench.ContactDataInjector</type>
    <description>inject data for Contact</description>
    <init-params>
     <value-param>
      <name>mA</name> <!-- maximum number of address books -->
      <value>5</value>
     </value-param>
     <value-param>
      <name>mC</name> <!-- maximum number of contacts per a address books -->
      <value>10</value>
     </value-param>
    </init-params>
   </component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `ContactDataInjector`
- **Set-method**: `addInjector`
- **Type**: `org.exoplatform.contact.service.bench.ContactDataInjector`

Parameters	Possible Values	Default Values	Description
mA	number	5	The maximum number of address books of the Address Book application in each injection.
mC	number	5	The maximum number of contacts of an

Parameters	Possible Values	Default Values	Description
			address book in each injection.

2. Execute the injector by the RESTful request as follows:

```
http://{domain}/{rest}/bench/inject/ContactDataInjector?mA=5&mC=10
```

In which:

- **{domain}**: The domain name of the server.
- **{rest}**: The name of eXo REST service.

CalendarDataInjector

The *CalendarDataInjector* plug-in is used to manage data injection in the Calendar application.

To use this plug-in, do as follows:

1. Add the following configuration to the *configuration.xml* file to register the plug-in to the *DataInjectorService* component:

```
<external-component-plugins>
   <target-component>org.exoplatform.services.bench.DataInjectorService</target-component>
   <component-plugin>
     <name>CalendarDataInjector</name>
     <set-method>addInjector</set-method>
     <type>org.exoplatform.calendar.bench.CalendarDataInjector</type>
     <description>inject data for Calendar</description>
     <init-params>
       <value-param>
         <name>mCt</name> <!-- maximum number of categories -->
         <value>5</value>
       </value-param>
       <value-param>
         <name>mEcat</name> <!-- maximum number of event categories -->
         <value>10</value>
       </value-param>
       <value-param>
```

```
        <name>mCal</name> <!-- maximum number of calendars -->
        <value>10</value>
      </value-param>
      <value-param>
        <name>mEv</name> <!-- maximum number of events -->
        <value>5</value>
      </value-param>
      <value-param>
        <name>mTa</name> <!-- maximum number of tasks -->
        <value>5</value>
      </value-param>
      <value-param>
        <name>typeOfInject</name> <!-- type of inject -->
        <value>all</value> <!-- string all/public/private -->
      </value-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `CalendarDataInjector`
- **Set-method**: `addInjector`
- **Type**: `org.exoplatform.calendar.bench.CalendarDataInjector`

Parameters	Possible Values	Default Values	Description
mCt	number	5	The maximum number of categories in each injection.
mEcat	number	10	The maximum number of event categories in each injection.
mCal	number	10	The maximum number of calendars in each injection.
mEv	number	5	The maximum number of events in one calendar in each injection.

Parameters	Possible Values	Default Values	Description
mTa	number	5	The maximum number of tasks in one calendar in each injection.
typeOfInject	String	all	The type of injection, including "public", "private" and "all". If the type is set as "public", only public calendars are created. If the type is set as "private", only personal calendars are create. If the type is set as "all", both the public and private ones are created.

2. Execute the injector by the RESTful request as follows.

```
http://{domain}/{rest}/bench/inject/
CalendarDataInjector?mCt=5&mEcat=10&mCal=10&mEv=20&mTa=20&typeOfInject=private
```

MailDataInjector

The *MailDataInjector* is used to manage data injection in the Mail application.

Because of the quite complicated architecture of the Mail application in eXo Collaboration, the injector uses a simple mail server (a mock server) to simulate the way a mail server works. This will create real mail data without mocking effort from the injector and create a reliable testing environment.

To use the injector, do as follows:

1. Add the Greenmail library to the classpath of the web server (in tomcat, copy it to the **lib** folder). The library is Greenmail 1.3.1b but including some bug fixes which are not updated in the counterpart version of Icegreen. (To have the library, contact eXo Support team).

2. Initialize this component as a service of GateIn, then it will be invoked by MailDataInjector.

```
<component>
```

```
    <type>org.exoplatform.mail.service.bench.SimpleMailServerInitializer</type>
</component>
```

3. Register *MailDataInjector* to *DataInjectorService* by the following configuration:

```
<external-component-plugins>
  <target-component>org.exoplatform.services.bench.DataInjectorService</target-component>
  <component-plugin>
      <name>MailDataInjector</name>
      <set-method>addInjector</set-method>
      <type>org.exoplatform.mail.service.bench.MailDataInjector</type>
      <description>inject data for Contact</description>
    </component-plugin>
</external-component-plugins>
```

In which:

- **Name**: `MailDataInjector`
- **Set-method**: `addInjector`
- **Type**: `org.exoplatform.mail.service.bench.MailDataInjector`

Usage of MailDataInjector

- To insert mail data into the Mail application, the request link is as follows:

```
http://{domain}/{rest}/bench/inject/MailDataInjector/
?users=mary,root&accounts=2,account&inPro=IMAP&check=true&msgs=100&attSize=100
```

Params	Values	Description
users	String	The list of users sepereated by commas.
accounts	String	The number of accounts injected by the data injector. This value consists of two parts seperated by commas: the first is the number of

Params	Values	Description
		accounts of an user, the second is the prefix of the account Id.
inPro	String	The incoming protocol. The possible values are: POP3 and IMAP.
check	Boolean	Specify if checking mails after the data are created on the mail server or not. If the value is true, the injector will execute checking new message tasks sequentially. This task can take some time if the data are large.
msgs	Number	The number of messages will be available in each account.
attSize	Number	The size of an attachment. If it does not exist or is equal to 0, no file is attached to the mail message.

- To reject mail data from the Mail application, the request link is as follows:

```
http://{domain}/{rest}/bench/reject/MailDataInjector/?users=root&accounts=2,account
```

This link will request for removing 2 accounts of "root" of which Id starts with "account".

Note

By default, such settings have been declared in "csdemo.war/WEB-INF/conf/csdemo/cs/bench-configuration.xml". Therefore, to save time, you should import the *bench-configuration.xml* file to "*csdemo.war/WEB-INF/conf/configuration.xml*", and then modify it as your purpose rather than create a new one.

eXo Chatserver Configuration

Openfire Configuration

The Chat service of eXo Collaboration is a Jabber engine powered by Openfire. eXo Platform will delegate the actual Jabber protocol communication to Openfire.

You have the full latitude to configure Openfire. There are two possible ways to do it:

- The admin console: *http://localhost:9090/.*
- The openfire.xml file in *$openfire_home/conf/*.

Configuration in Openfire.xml

The Openfire server has a single configuration file called *openfire.xml* and located under the *exo-openfire/conf* directory. Configuration is based on properties expressed in an XML syntax. For example, to set the property prop.name.is.blah=value, you would write this xml snippet:

```
<prop>
  <name>
    <is>
      <blah>value</blah>
    </is>
  </name>
</prop>
```

Openfire has an extensive list of configuration properties. You can read a list of all properties on this page: *http://www.igniterealtime.org/community/docs/DOC-1061.*

eXo specific configuration

The eXo Collaboration bundle comes with a pre-configured Openfire server. It is bundled with some eXo plugins and configurations that allow connectivity with eXo Platform. The key properties for integration are:

- *provider.auth.className*: An implementation of the AuthProvider interface for authentication of users on the Chat server.
- *provider.users.className*: An implementation of the UserProvider interface to which Openfire will delegate users management.
- *provider.groups.className*: An implementation of the GroupProvider interface to which Openfire will delegate groups management.

eXo Platform provides implementations for these 3 interfaces with ExoAuthProvider, ExoUserProvider, ExoGroupProvider. These implementations are based on the eXo REST

framework and let you configure the endpoints within the *openfire.xml* file with additional properties:

Property	Default value	Description
eXo.env.serverBaseURL	`http://localhost:8080/#`	The base URL of the server.
eXo.env.restContextName	`rest`	The context name of REST Web application.
provider.authorizedUser.name	`root`	The username to authenticate to access the HTTP REST service.
provider.authorizedUser.password	`gtn`	The password matching with provider.authorizeduser.name.
exoAuthProvider.authenticationURL	`/organization/authenticate/`	The URL to authenticate users.
exoAuthProvider.authenticationMethod	`POST`	The HTTP method used for the authentication method.
exoUserProvider.findUsersURL	`/organization/xml/user/find-all/`	The URL to find all users.
exoUserProvider.findUsersMethod	`GET`	The HTTP method used to find all users in the system.
exoUserProvider.getUsersURL	`/organization/xml/user/view-range/`	The URL to retrieve a range of users.
exoUserProvider.getUsersMethod	`GET`	The HTTP method used for user/view-range.
exoUserProvider.usersCountURL	`/organization/xml/user/count`	The URL to count the number of users.
exoUserProvider.usersCountMethod	`GET`	The HTTP method used to count the number of users.
exoUserProvider.userInfoURL	`/organization/xml/user/info/`	The URL to get the information of users.
exoUserProvider.userInfoMethod	`GET`	The HTTP method used to get the information of users.
exoGroupProvider.groupInfoURL	`/organization/xml/group/info/`	The URL to get the information of a group of users.
exoGroupProvider.groupInfoMethod	`GET`	The HTTP method used to get the information of a group of users.
exoGroupProvider.getGroupsAllURL	`/organization/xml/group/view-all/`	The URL to view a list of all user groups.

Property	Default value	Description
exoGroupProvider.getGroupsAllMethod	GET	The HTTP method used to view a list of all user groups.
exoGroupProvider.getGroupsRangeURL	/rest/organization/xml/group/view-from-to/	The URL to list groups in a specific range.
exoGroupProvider.getGroupsRangeMethod	GET	The HTTP method used to list groups in a specific range.
exoGroupProvider.getGroupsForUserURL	/rest/organization/xml/group/groups-for-user/	The URL to list groups to which a user belongs.
exoGroupProvider.getGroupsForUserMethod	GET	The HTTP method used to list groups to which a user belongs.
exoGroupProvider.groupsCountURL	/rest/organization/xml/group/count	The URL to count the number of groups.
exoGroupProvider.groupsCountMethod	GET	The HTTP method used to count the number of groups.

As you can see, the default settings will only work if eXo Platform is deployed on the same host as Openfire, on the port 8080.

Note

restContextName is used to specify the Openfire server that is dedicated for the portal. If the *eXo.env.restContextName* system property exists, it will override this value.

The *eXo.env.restContextName* system property can be set by specifying the -D option to the Java command when running Openfire.

For example:

- If the Openfire server is dedicated for the portal named "portal", the command will have the following format: *java -DeXo.env.restContextName=rest -jar ../lib/startup.jar* .
- If the Openfire server is dedicated for the portal named "csdemo", the command will have following format: *java -DeXo.env.restContextName=rest-csdemo -jar ../lib/startup.jar.*.

By default, the Openfire server is dedicated to the portal named "portal".

System Configuration

Openfire makes use of several ports for communication.

Interface	Port	Type	Description
All addresses	`5222`	`Client to Server`	The standard port for clients is to connect to the server. Connection may or may not be encrypted. You can update the security settings for this port.
All addresses	`9090 & 9091`	`Admin Console`	The ports used for the unsecured and secured Openfire Admin Console accesses respectively.
All addresses	`7777`	`File Transfer Proxy`	The port used for the proxy service that allows files to be transferred between two entities on the XMPP network.
All addresses	`3478 & 3479`	`STUN Service`	The port used for the service that ensures connectivity between entities behind a NAT.

You can view the table above in *http://hostname:9090/index.jsp* after you have logged into the Openfire's web console and also customize those ports by yourself.

AS configuration

To enable the propagation of identity across the Chat webapp, you are required to enable the SSO valve on the Tomcat-based Application server.

- For the Jboss server, edit *jboss/server/default/deploy/jboss-web.deployer/server.xml*.
- For the Tomcat server, edit *tomcat/conf/server.xml*.

The valve should already be there, you just need to uncomment it if it is not already done.

```
<Valve className="org.apache.catalina.authenticator.SingleSignOn"/>
```

In case of the cluster deployment, you may want to use `ClusteredSingleSignOn` instead.

```
<Valve className="org.jboss.web.tomcat.service.sso.ClusteredSingleSignOn"/>
```

JCR Structure

eXo Collaboration is a JCR-based product, so data of eXo Collaboration are managed by the eXo-JCR service with each specific structure. The chapter aims at outlining the JCR structure of each application in eXo Collaboration through diagrams and then describing properties of main node types.

Each diagram shows nodes and their primary node types. Every node/child node must have only one primary node type represented in the round bracket () under the node/childnode, but may also have many mixin node types. Because mixin nodes cannot define the node structure like the primary nodes, they are not shown in the diagrams and their properties hereafter are not described.

Note

To learn more about the eXo Collaboration JCR Structure, you should have the certain knowledge of *JCR* [http://jcp.org/en/jsr/detail?id=170].

Calendar JCR Structure

The Calendar data are saved in eXo-JCR under the CalendarApplication data directory. The Calendar JCR Structure is divided into two main branches: one for public (exo:application) and the other for users (Users).

The whole JCR structure of Calendar can be visualized in the diagram below:

calendars

The **Calendars** node of the nt:unstructured type contains the child nodes of the **exo:calendar** type. When a calendar is created by users or the default ones in the system, it is stored under the **calendars** node: *CalendarApplication/calendars/%calendar_id%*. Its node type is *exo:calendar* that has the following properties:

Property name	Required type	Multiple	Description
exo:id	String	false	The Id of the calendar.
exo:name	String	false	The name of the calendar.
exo:description	String	false	The brief description of the calendar.
exo:viewPermissions	String	true	The list of users/ groups having the view permissions.

Property name	Required type	Multiple	Description
exo:editPermissions	String	true	The list of users/ groups having the edit permissions.
exo:groups	String	true	The list of user groups to which the calendar belongs.
exo:categoryId	String	false	The Id of the category containing the calendar.
exo:calendarColor	String	false	The color name of the calendar that is defined in the *org.exoplatform.web ui.form.ext.UIFormColorPicker class (such as Sky blue, Powder blue).*
exo:calendarOwner	String	false	The name of the user creating the calendar.
exo:locale	String	false	Location where the calendar is set in format of the uppercase ISO 3166 3-letter country code.
exo:timeZone	String	false	The Id of the time zone that is set by the user in compliance with the Java class: java.util.TimeZone.
exo:publicUrl	String	false	The public ICAL link of the calendar.
exo:privateUrl	String	false	The private ICAL link of the calendar.

When a user shares his own calendar with other users, the Id of the calendar node is referred to the node under the **sharedCalendar** node: *CalendarApplication/sharedCalendars/%user_id%* following the JCR reference mechanism.

In case of users' private calendar, two mixin node types *exo:remoteCalendar* and *exo:calendarShared* can be added to the *exo:calendar* node type.

- The *exo:remoteCalendar* mixin node type has the following properties:

Property name	Required type	Multiple	Description
exo:remoteUrl	String	false	The URL of the remote calendar.
exo:remoteType	String	false	The type of the remote calendar, including ICalendar (.ics) and CalDav.
exo:username	String	false	The username used to access the remote calendar.
exo:password	String	false	The password used to access the remote calendar.
exo:syncPeriod	String	false	The period the remote calendar is synchronized. auto, 5 minutes, 10 minutes, 15 minutes, 1 hour, 1 day, 1 year
exo:lastUpdated	Date	false	The last update of the remote calendar.
exo:beforeDate	String	false	The period before the current date in which the calendar is checked out, including the values: None (the unlimited time), 1 week, 2 weeks, 1month, 2 months, 3 months, 6 months and 1 year.
exo:afterDate	String	false	The period after the current date in which the calendar is checked out, including the values: Forever (the unlimited time), 1 week, 2 weeks, 1month, 2 months, 3 months, 6 months and 1 year.

- The *exo:calendarShared* mixin node type has the following properties:

Property name	Required type	Multiple	Description
exo:sharedId	Reference	true	The user Ids who are shared the calendars.

An event can have many attachments which are stored under the **attachment** node of the *exo:eventAttachment*type: *CalendarApplication/calendars/%calendar_id%/%event_id%/attachment/%attachment_id%*. The *exo:eventAttachment* node type has the following properties:

Property name	Required type	Multiple	Description
exo:fileName	String	false	The name of the attached file.

eventCategories

The **eventCategories** node contains all event categories. When an event category is created, it is stored in a node of the *exo:eventCategory* type, under the **eventCategories** node defined at the path: CalendarApplication/eventCategories/%eventcategory_id%.

This node type has the following properties:

Property name	Required type	Multiple	Description
exo:id	String	false	The Id of the category to which an event belongs.
exo:name	String	false	The name of the category to which an event belongs.
exo:description	String	false	The brief description of the category to which an event belongs.

Each event category node contains the calendar event node of the *exo:calendarEvent)* type. This node of the *exo:calendarEvent* type is stored at the path: *CalendarApplication/eventCategories/%eventcategory_id%/%event_id%*.

This node type has the following properties:

Property name	Required type	Multiple	Description
exo:id	String	false	The Id of the event.
exo:eventType	String	false	Type of the event, including Event and Task.
exo:summary	String	false	

Property name	Required type	Multiple	Description
			The summary of the event.
exo:location	String	false	The location where the event will take place.
exo:taskDelegator	String	false	The name of the user being delegated the task.
exo:description	String	false	The brief description of the event.
exo:eventCategoryId	String	false	The Id of the category containing the event.
exo:eventCategoryName	String	false	The name of the category containing the event.
exo:calendarId	String	false	The Id of the calendar containing the event.
exo:fromDateTime	Date	false	The start time of the event.
exo:toDateTime	Date	false	The end time of the event.
exo:priority	String	false	The preference order of the event, including 4 values: none, low, normal, high.
exo:isPrivate	Boolean	false	Define if the event is private or not.
exo:eventState	String	false	The state of the event which depends on each event type.
exo:invitation	String	true	The list of email addresses of users being invited to the event. This property is for the Event type only.
exo:participant	String	true	The list of users being invited to the event.

Property name	Required type	Multiple	Description
			This property is for the Event type only.
exo:participantStatus	true	String	The status of the participant, including name and status value.
exo:message	String	false	The content of the invitation email.
exo:repeat	String	false	Repetition type of the event, including: "norepeat", "daily", "weekly", "monthly", "yearly", "weekend", "workingdays".
exo:sendOption	String	false	The option to notify users before sending the invitation via email: never (not sending all time), always (sending without asking) and ask (asking before sending).

categories

The **categories** node of the *nt:unstructured* type contains the child nodes of the *exo:calendarCategory* type. These child nodes containing the Id of the calendars in the categories are stored under the **categories** node: *CalendarApplication/categories/%calendarcategories_id%*.

The *exo:calendarCategory* node type has the following properties:

Property name	Required type	Multiple	Description
exo:id	String	false	The Id of the category to which a calendar belongs.
exo:name	String	false	The name of the category to which a calendar belongs.
exo:description	String	false	The brief description of the category to

Property name	Required type	Multiple	Description
			which a calendar belongs.
exo:calendarIds	String	true	A list of calendar Ids belonging to the category.

eXoCalendarFeed

The **eXoCalendarFeed** of the *nt:unstructured* type contains **iCalendars**, **webDavCalendars** as child nodes and others of the *exo:rssData* type.

The *exo:rssData* node type has the following properties:

Property name	Required type	Multiple	Description
exo:baseUrl	String	false	The original link to the RSS source.
exo:title	String	false	The title of the feed.
exo:content	Binary	false	The content of the feed.

The **iCalendars** node of the *nt:unstructured* type contains the child nodes of *exo:iCalData* type.

The *exo:iCalData* node type has the following properties:

Property name	Required type	Multiple	Description
exo:data	Binary	false	The exported content of the calendar in the ics.format.

The **webDavCalendars** node of the *nt:unstructured* type contains the child nodes of the *exo:caldavCalendarEvent* type.

The *exo:caldavCalendarEvent* node type has the following properties:

Property name	Required type	Multiple	Description
exo:caldavHref	String	false	The URL of the remote calendar event.
exo:caldavEtag	String	false	The tag of the remote calendar event.

Y%yyyy%

The **Y%yyyy%** of the *nt:unstructured* type has the name beginning with the Y character followed by the year name having 4 numbers. It contains all the child nodes of **M%mm%**.

The **M%mm%** of the *nt:unstructured* type has the name beginning with the M character followed by the month name having 2 numbers. It contains all the child nodes of **D%dd%**.

The **D%dd%** of the *nt:unstructured* type has the name beginning with the D character followed by the date having 2 numbers. This node has two child nodes: **reminder** and **events**.

The **reminder** node of the *nt:unstructured* type contains the child nodes named basing on the Id of the event. This child node also has the nt:unstructured type. Each node is used to classify reminders of the same event. Each reminder is stored under a node of the *exo:reminder* type: *CalendarApplication/Y%yyyy%/M%mm%/D%dd%/reminders/%event_id%/%reminder_id%*.

The *exo:reminder* node type has the following properties:

Property name	Required type	Multiple	Description
exo:id	String	false	The Id of the reminder.
exo:eventId	String	false	The event Id of the reminder.
exo:creator	String	false	Define who creates the reminder.
exo:alarmBefore	Long	false	The amount of time that the reminder message is sent before the event starts.
exo:email	String	false	The list of emails to which the reminder message is sent.
exo:timeInterval	Long	false	Interval for resending the reminder message in minutes.
exo:reminderType	String	false	The types of reminders, including email and pop-up.
exo:fromDateTime	Date	false	The start time to send the reminder.
exo:remindDateTime	Date	false	The time to send the reminder.
exo:isRepeat	Boolean	false	Check if the reminder is repeated or not.
exo:isOver	Boolean	false	Check if the reminder is expired or not.
exo:summary	String	false	The summary of the reminder.

Property name	Required type	Multiple	Description
exo:description	String	false	The brief description of the reminder.

The **events** node of the *nt:unstructured* type contains the child node of the *exo:calendarPublicEvent* type defined at the path: *CalendarApplication/Y%yyyy%/M%mm%/D%dd%/events/%event_id%*.

Property name	Required type	Multiple	Description
exo:id	String	false	The Id of the public event.
exo:eventType	String	false	Event type, including Task and Event.
exo:calendarId	String	false	The calendar Id of the public event.
exo:rootEventId	String	false	The Id of each corresponding node: exo:calendarEvent.
exo:fromDateTime	Date	false	The start time of the public event.
exo:toDateTime	Date	false	The snd time of the public event.
exo:participant	String	true	The list of users being invited to the public event.
exo:eventState	String	false	The state of the public event, including: busy, available, outside.

The **events** node can add the **exo:repeatCalendarEvent** mixin node that has the following properties:

Property name	Required type	Multiple	Description
exo:repeatCount	Long	false	The number of times that the event is repeated.
exo:repeatUntil	Date	false	The given time until when the event is repeated.
exo:repeatInterval	Long	false	The interval when the event is repeated. It can be day,

Property name	Required type	Multiple	Description
			week, month or year corresponding to the repetition type chosen of day, week, month or year.
exo:repeatByDay	String	true	The given days in a week on which the event is repeated.
exo:repeatByMonthDay	Long	true	The given day/date in a month on which the event is repeated.
exo:recurrenceId	String	false	The Id of each event in the event series.
exo:excludeId	String	true	The Id of the events that are removed from the event series.
exo:isException	Boolean	false	Show whether the event is the exception in the event series or not. This case occurs when the event is removed from the repeated event series.
exo:originalReference	Reference	false	The UUID of the event that is repeated first.
exo:repeatFinishDate	Date	false	The end date on which the event is repeated.

calendarSetting

The **calendarSetting** node of the *exo:calendarSetting* type is stored in *CalendarApplication/calendarsetting*. The *exo:calendarSetting* node type has the following properties:

Property name	Required type	Multiple	Description
exo:viewType	String	false	View type of the calendar. For more details, refer to the *org.exoplatform.calendar.service.CalendarSetting class.*

Property name	Required type	Multiple	Description
exo:timeInterval	Long	false	The interval for each action displayed each UI, for example, dragging and dropping one event in the Calendar application.
exo:weekStartOn	String	false	Define the start date of one week, complying with the *org.exoplatform.calendar.service.CalendarSetting class.*
exo:dateFormat	String	false	Define the date format, including dd/MM/yyyy, dd-MM-yyyy, MM/dd/yyyy, and MM-dd-yyyy.
exo:timeFormat	String	false	Define the time format, including "hh:mm a" and "HH:mm".
exo:location	String	false	Location where the calendar is set in format of the uppercase ISO 3166 3-letter country code.
exo:timeZone	String	false	The Id of the time zone, which is set by the user in compliance with the Java class: java.util.TimeZone.
exo:showWorkingTime	false	Boolean	Check if the working period is displayed or not.
exo:workingTimeBegin	String	false	Time to start working. This property only takes effect when exo:showWorkingTime is set to true.

Property name	Required type	Multiple	Description
exo:workingTimeEnd	String	false	Time to end working. This property only takes effect when exo:showWorkingTime is set to true.
exo:defaultPrivateCalendars	String	true	The list of the hidden private calendars.
exo:defaultPublicCalendars	String	true	The list of the hidden public calendars.
exo:defaultSharedCalendars	String	true	The list of the hidden shared calendars.
exo:sharedCalendarsColors	String	true	Define the color of the shared calendar, which is in the format of *calendar id*:*color name*.
exo:sendOption	String	false	The option to notify users before sending an invitation via email: never (not sending all time), always (sending message without asking) and ask (asking before sending).

Chat JCR Structure

The node type **lr:conversation** has the following properties:

Property name	Required type	Description
lr:conversationstartDate	Date	Start date of the conversation.
lr:conversationlastActiveDate	Date	Last date when the conversation is updated.

The node type **lr:historicalmessage** has the following properties:

Property name	Required type	Description
lr:messagefrom	String	

Property name	Required type	Description
		Jabber Id of the user (or chat room) sending (or containing) the message respectively.
lr:messageto	String	Jabber Id of the user (or chat room) to whom (to which) the message is sent.
lr:messagetype	String	List of message types. For more details, refer to the *org.jivesoftware.smack.packet.Message.Type class.*
lr:messagebody	String	Main content of the message.
lr:messagedateSend	Date	Date when the message was sent.
lr:messagereceive	Boolean	Check if the message has been received or not.

The node type **lr:participantchat** has the following properties:

Property name	Required type	Description
lr:participantchatjid	String	Jabber Id of the user.
lr:participantchatusername	String	Username of the portal.

The node type **lr:interlocutor** contains information regarding to the conversation between two users or of the chat room. It has the following properties:

Property name	Required type	Description
lr:conversationId	String	Id of the conversation which is the JCR node name of lr:conversation.
lr:interlocutorjid	String	Jabber Id of the chat room or user.
lr:interlocutorname	String	Username or name of the chat room.
lr:interlocutorisRoom	Boolean	Define if the conversation is performed between two users or is of chat room.

The node type **lr:defaultpresencestatus** has the following properties:

Property name	Required type	Description
lr:conversationlastActiveDate	Date	

Property name	Required type	Description
		Date when the conversation is last updated.

The node type **lr:presencestatus** contains information regarding to the current status of user. It has the following properties:

Property name	Required type	Description
lr:userid	String	Id of the user.
lr:status	String	Current status of the user included in the *org.jivesoftware.smack.packet.Presence.Type class.*

Address Book JCR Structure

The Address Book portlet is a JCR-based application. The Address Book data are stored in the eXo-JCR under the ContactApplication data directory. The whole JCR structure of Address Book can be visualized in the diagram below:

Contacts

The **Contacts** node of the *nt:unstructured* type has the child nodes of the *exo:contact*. When a contact is created and added to an address book, it is stored in a node defined at the path: *ContactAppication/Contacts/%contact_id%*. When a contact is tagged, it is saved in the **tags** node with the *exo:tags* property. In case, a contact is shared to other users, it is referred to the **SharedContact** node of the *exo:sharedId* type. The *exo:contact* node type has the following properties:

Property name	Required type	Multiple	Description
exo:id	String	false	Node name of the exo:contact property.
exo:fullName	String	false	The full name of the contact.
exo:firstName	String	false	The first name of the contact.
exo:lastName	String	false	The last name of the contact.
exo:nickName	String	false	The nickname of the contact.
exo:gender	String	false	The gender of the contact.

Property name	Required type	Multiple	Description
exo:birthday	Date	false	The birthday of the contact.
exo:jobTitle	String	false	The job title of the contact.
exo:emailAddress	String	true	The email address of the contact.
exo:exoId	String	false	The Id of the user in the Chat application of eXo Collaboration.
exo:googleId	String	false	The Google Id of the user.
exo:msnId	String	false	The MSN Id of the user.
exo:aolId	String	false	The AOL Id of the user.
exo:yahooId	String	false	The Yahoo Id of the user.
exo:icrId	String	false	The ICR Id of the user.
exo:skypeId	String	false	The Skype Id of the user.
exo:icqId	String	false	The ICQ Id of the user.
exo:homeAddress	String	false	The home address of the contact.
exo:homeCity	String	false	The home city of the contact.
exo:homeState_province	String	false	The home state/ province of the contact.
exo:homePostalCode	String	false	The home postal code of the contact.
exo:homeCountry	String	false	The home country of the contact.
exo:homePhone1	String	false	The primary home phone number of the contact.
exo:homePhone2	String	false	The secondary home phone number of the contact.

Property name	Required type	Multiple	Description
exo:homeFax	String	false	The home fax of the contact.
exo:personalSite	String	false	The personal site of the contact.
exo:workAddress	String	false	The address where the contact works.
exo:workCity	String	false	The city where the contact works.
exo:workState_province	String	false	The state/province where the contact works.
exo:workPostalCode	String	false	The postal code of the location where the contact works.
exo:workCountry	String	false	The country where the contact works.
exo:workPhone1	String	false	The primary phone number at the contact's working location.
exo:workPhone2	String	false	The secondary phone number at the contact's working location.
exo:workFax	String	false	The fax number at the contact's working location.
exo:mobilePhone	String	false	The mobile phone number of the contact.
exo:webPage	String	false	The website of the contact.
exo:note	String	false	The note about the contact.
exo:categories	String	true	The list of categories created by the user.
exo:editPermissionUsers	String	true	The list of users obtaining the edit permission.

Property name	Required type	Multiple	Description
exo:viewPermissionUsers	String	true	The list of users obtaining the view permission.
exo:editPermissionGroups	String	true	The list of groups obtaining the edit permission.
exo:viewPermissionGroups	String	true	The list of groups obtaining the view permission.
exo:tags	String	true	The list of tag Ids which the contact has marked.
exo:lastUpdated	Date	false	The time when the contact is last updated.
exo:isOwner	Boolean	false	Show if this contact is the contact of the user in the system or not.
exo:ownerId	String	false	If the contact is a user in the system, the owner Id will be the username of this user.

This node type may add the *exo:contactShared* mixin that has the following properties:

Property name	Required type	Multiple	Description
exo:sharedUserId	String	false	The name of the user sharing the contact.
exo:sharedId	Reference	true	The list of the references to shared users/groups.s

ContactGroup

The **ContactGroup** node of the *nt:unstructured* type contains all the child nodes of the *exo:contactGroup* type. These child nodes store address books and are stored in *ContactApplication/ContactGroup/%addressbook_id*. This node is also referred to the *exo:contact* node type with the *exo:categories* property. When an address book is shared, it is referred to the **SharedAddressBook** node with the *exo:sharedId* property.

The information of each address book is contained in a node of the *exo:contactGroup* type that has the following properties:

Property name	Required type	Multiple	Description
exo:id	String	false	The Id of the address book.
exo:name	String	false	The name of the address book.
exo:description	String	false	The brief description of the address book.
exo:editPermissionUsers	String	true	The list of users having the permission to edit the address book.
exo:viewPermissionUsers	String	true	The list of users having the permission to view the address book.
exo:editPermissionGroups	String	true	The list of groups having the permission to edit the address book.
exo:viewPermissionGroups	String	true	The list of groups having the permission to view the address book.

tags

Tags are used to categorize the contacts in groups, so users can easily find the contacts. The **tags** node of the *nt:unstructured* type contains the child nodes of the *exo:contactTag* type. When a new tag is created, it is stored in a node defined at the path: *ApplicationData/ContactApplications/tags/%tags_id%*. The information of each tag is contained in a node of the *exo:contactTag* type that has the following properties:

Property name	Required type	Multiple	Description
exo:id	String	false	The Id of the tag.
exo:name	String	false	The name of the tag.
exo:description	String	false	The brief description of the tag.
exo:color	String	false	The color of the tag.

Shared

When a contact or a address book is shared to other users, it is referred to the **SharedContact** or **SharedAddressBook** node. Both of these child nodes have the *nt:unstructured* type.

Mail JCR Structure

The node type **exo:account** has the following properties:

Property name	Required type	Description
exo:id	String	Id of the account.
exo:label	String	Name of the account.
exo:userDisplayName	String	Screen name of the user.
exo:emailAddress	String	Email address of the account.
exo:emailReplyAddress	String	Email address of the account receiving replies.
exo:signature	String	Signature of the account.
exo:description	String	Brief description of the account.
exo:checkMailAuto	Boolean	Define if the mail is automatically checked after a given period or not.
exo:emptyTrash	Boolean	Define if the trash needs to be cleaned up when exiting from the Mail application or not.
exo:serverProperties	String	Information of the POP/IMAP server configuration.
exo:smtpServerProperties	String	Information of the SMTP server configuration.
exo:lastCheckedTime	Date	Time when the account was last checked.
exo:checkAll	Boolean	Define if all folders of the mail are checked or not.
exo:checkFromDate	Date	Get mails as from the given date only if the value of exo:serverProperties is set for configuring the IMAP server.
exo:isSavePassword	Boolean	

Property name	Required type	Description
		Define if the password is saved or not.
exo:secureAuthsIncoming	String	Type of the incoming connection for security. Its values include starttls, ssl/tls.
exo:secureAuthsOutgoing	String	Type of the outgoing connection for security. Its values include starttls, ssl/tls.
exo:authMechsIncoming	String	Authentication mechanism of the incoming connections. Its values consist of ntlm, plain, login, digest-md5, kerberos/ gssapi, cram-md5.
exo:authMechsOutgoing	String	Authentication mechanism of the outgoing connections. Its values consist of ntlm, plain, login, digest-md5, kerberos/ gssapi, cram-md5.
exo:permissions	String	Permissions of delegators.

The node type **exo:folder** has the following properties:

Property name	Required type	Description
exo:id	String	Id of the folder.
exo:name	String	Name of the folder.
exo:label	String	Absolute path referring to the folder on the Mail server.
exo:unreadMessages	Long	Number of unread messages in the folder.
exo:totalMessages	Long	Total number of messages in the folder.
exo:personal	Boolean	Define if the folder is created by one user or the Mail system.
exo:folderType	Long	Type of folder, which is defined in the javax.mail.Folder class.
exo:lastStartCheckingTime	Date	Start time of the last check in the folder.
exo:lastCheckedTime	Date	End time of the last check in the folder.

The node type **exo:message** has the following properties:

Property name	Required type	Description
exo:id	String	Id of the message.
exo:uid	String	Id of the message on the IMAP server.
exo:inReplyToHeader	String	Id of the first message in the matching thread.
exo:path	String	Absolute path of the exo:message type.
exo:account	String	Id of the account.
exo:from	String	Value given in the **From** field in the email message, containing information of the sender, such as full name and email.
exo:to	String	Value given in the **To** field in the email message, containing information of the receiver, such as full name and email.
exo:cc	String	Value given in the **CC** field in the email message, containing information of the receivers, such as full name and email.
exo:replyto	String	Value given in the **Reply-To** field in the email message, such as emails.
exo:isUnread	Boolean	Define if the email has been read or not.
exo:subject	String	Subject of the email message that can be read from the **Subject** field.
exo:body	String	Main content of the email message.
exo:sendDate	Date	Date when the email message was sent.
exo:receivedDate	Date	Date when the email message was received.
exo:size	Long	Capacity of the email message in bytes.
exo:contentType	String	

Property name	Required type	Description
		Content type of the email message, for example: text/plain and text/html.
exo:folders	String	List of folder Ids containing the email message.
exo:tags	String	List of tag Ids marked in the email message.
exo:star	Boolean	Define if the email message is starred or not.
exo:hasAttach	Boolean	Define if any files are attached with the email message or not.
exo:priority	Long	Preference order of the message with 3 default values: 1 = High, 3 = Normal, 5 = Low.
exo:lastUpdateTime	Date	Time when the message was last updated.

The node type **exo:mailAttachment** has the following property:

Property name	Required type	Description
exo:fileName	String	Name of the file attached in the mail.

The node type **exo:mailtag** has the following properties:

Property name	Required type	Description
exo:id	String	Tag id of the mail.
exo:name	String	Name of the tag.
exo:description	String	Brief description of the mail tag.
exo:color	String	Color of the tag which is defined in the *org.exoplatform.webui.form.ext.UIFormColorPicke* class.

The node type **exo:filter** has the following properties:

Property name	Required type	Description
exo:id	String	Filter id which is a unique and randomized value.
exo:name	String	Name of the filter which is defined by the user.

Property name	Required type	Description
exo:from exo:to exo:subject exo:body	String	Filter email messages by each field respectively: * From * To * Subject * Body
exo:fromCondition exo:toCondition exo:subjectCondition exo:bodyCondition	Long	Filter emails by the condition types set in each property respectively: * exo:from * exo:to * exo:subject * exo:body All these properties have two values: * 0 = returned messages contains the value set in the corresponding property. * 1 = do not contain the value set in the corresponding property.
exo:applyTag	String	Apply the tag for the filtered email messages.
exo:applyFolder	String	Apply the folder for the filtered email messages.

Property name	Required type	Description
exo:keepInbox	Boolean	Define if the email message is still kept in the Inbox folder or not.
exo:applyForAll	Boolean	If the value is set to "true" into the exo:applyForAll property, the filter will be executed for all email messages.

The node type **exo:mailSetting** has the following properties:

Property name	Required type	Description
exo:numberMsgPerPage	Long	Number of messages displayed in one page.
exo:formatAsOriginal	Boolean	Define if the email message got from the mail server is kept in the original format or not.
exo:replyWithAttach	Boolean	Make the original message as the attachment before replying or not.
exo:forwardWithAttach	Boolean	Make the original message as the attachment before forwarding or not.
exo:prefixMsgWith	String	Prefix for the message.
exo:periodCheckAuto	Long	Time interval to check the email messages automatically.
exo:defaultAccount	String	Id of the user account that is displayed by default when the user logged in the Mail application.
exo:useWysiwyg	String	Define the Wysiwyg editor is used or not.
exo:saveMsgInSent	Boolean	Define the sent email message is saved to the Sent folder or not.
exo:layout	Long	Type of layout which is displayed to the user.
exo:returnReceipt	Long	Action type of the user when receiving the "return receipt" to confirm the arrival of one email

Property name	Required type	Description
		message, including: 0 = ask, 1 = never, 3 = always.

RSS JCR Structure

The node type **exo:content** has the following properties:

Property name	Required type	Description
id	String	Id of the content.
ownerType	String	Type of the owner. Its default value is user.
ownerId	String	User Id of owner.
dataType	String	Type of data.
data	String	XML string of the content navigation.
createdDate	Date	Created date of the content.
modifiedDate	Date	Modified date of the content.

Developer reference

Extension points

There are some overridable components in eXo Collaboration so that you can control how these components work by implementing or extending default implementations and then reconfigure these new components in the *configuration.xml* file.

ContentDAO

ContentDAO is an overridable component used in the Content application (or called RSS reader) of eXo Collaboration.

You can find the configuration file at *WEB-INF/cs-extension/cs/content/content-service-configuration.xml* with the declaration as below:

```
<component>
  <key>org.exoplatform.content.service.ContentDAO</key>
  <type>org.exoplatform.content.service.impl.ContentDAOImpl</type>
</component>
```

The example below is an example of plugin configuration:

```
<external-component-plugins>
  <target-component>org.exoplatform.content.service.ContentDAO</target-component>
  <component-plugin>
    <name>rssreader.listener</name>
    <set-method>addPlugin</set-method>
    <type>org.exoplatform.content.service.RSSContentPlugin</type>
    <description>rss reader plugin</description>
  </component-plugin>

  <component-plugin>
    <name>description.listener</name>
    <set-method>addPlugin</set-method>
    <type>org.exoplatform.content.service.DescriptionPlugin</type>
    <description>Description plugin</description>
  </component-plugin>
</external-component-plugins>
```

ContactLifeCycle

ContactLifeCycle is an interface to extend the capabilities of eXo Platform. The *ContactLifeCycle* lets you be notified during the lifecycle of an address book's contact when a contact is added or modified.

An example of *ContactLifeCycle* has been implemented to integrate the Address Book application in the eXo Social's Spaces. See the following configuration at: *ext/social-integration/src/main/resources/conf/portal/configuration.xml.*

See the following example:

```
<external-component-plugins>
  <target-component>org.exoplatform.contact.service.ContactService</target-component>
  <component-plugin>
    <name>ContactEventListener</name>
    <set-method>addListenerPlugin</set-method>
    <type>org.exoplatform.cs.ext.impl.ContactSpaceActivityPublisher</type>
  </component-plugin>
</external-component-plugins>
```

Details: `ContactSpaceActivityPublisher` implements `ContactLifeCycle`. This implementation publishes activities in the space activity stream to notify of new and updated contacts in the space address book.

Transport

Transport is an overridable component used in the Chat application of eXo Collaboration.

This overridable component is used to help users add the protocol to the Chat application, such as: ICQ, YAHOO, MSN, XMPP, AIM, GTALk.

The Chat application of eXo Collaboration only uses the XMPP protocol that is implemented in the *XMPPTransport* object.

EventLifeCycle

EventLifeCycle is an extension point used in the Calendar application of eXo Collaboration. You can find the configuration file of this component at: *ext/social-integration/src/main/resources/conf/portal/configuration.xml*.

See the following example:

```
<external-component-plugins>
  <target-component>org.exoplatform.calendar.service.CalendarService</target-component>
  <component-plugin>
    <name>CalendarEventListener</name>
    <set-method>addEventListenerPlugin</set-method>
    <type>org.exoplatform.cs.ext.impl.CalendarSpaceActivityPublisher</type>
  </component-plugin>
</external-component-plugins>
```

Details: `CalendarSpaceActivityPublisher` implements `EventLifeCycle`. It writes activities in the space activity stream when events or tasks are added/modified.

Public REST APIs

eXo Collaboration implements and provides many public REST APIS to help built-in applications, such as Calendar, Chat and Mail to communicate and transfer data with the server. By using these public REST APIs, extended or 3rd party applications can make use of this to develop cool web applications faster without much implementation.

Calendar application

The Calendar application of eXo Collaboration uses CalendarWebservice to provide all APIs for working with calendars, such as creating personal/group calendars, sharing calendars, managing events/tasks.

The REST API of Calendar portlet is exposed by *org.exoplatform.services.cs.calendar.CalendarWebservice* class.

Service name	Service URL	Location	Description
CalendarWebservice	`$portalname/ $restcontextname/ private/cs/ calendar`	- Maven groupId: org.exoplatform.cs - ArtifactId: exo.cs.web.webservice	Call extended services of the Calendar application.

Details:

- $portalname: The name of the portal
- $restcontextname: The context name of rest webapplication which is deployed to the "$portalname" portal.
- Private: is optional and used for the protected access only. Such calls will require authentication.
- **APIs usage**:

Use the following APIs to build all functions of the Calendar application:

Name	Service URL endpoint	Parameters	Expected Values	Description
checkpermission	/private/cs/calendar/checkPermission/{username}/{calendarId}/type}	username calendarid type	user id calendar id INVALID_TYPE = -1 / PRIVATE_TYPE = 0 / SHARED_TYPE = 1 / PUBLIC_TYPE = 2	Check the permission of a user to a calendar, aiming at defining if the user has the edit permission to the calendar or not.
event	/private/cs/calendar/event/{username}/{eventFeedName}	username eventFeedName	user id string	Return a feed RSS that lists links to access a specific event.
feed	/private/cs/calendar/feed/{username}/{feedname}/{filename}	username feedname filename	username string string	Show the content of a feed that is a list of events in the "filename" file.
publicProcess	/cs/calendar/subscribe/{username}/{calendarId}/{type}	username calendarId type	user id calendar id INVALID_TYPE = -1 / PRIVATE_TYPE = 0 / SHARED_TYPE = 1 / PUBLIC_TYPE = 2	Process the public calendar when having a remote access request.
privateProcess	/private/cs/calendar/private/{username}/	username calendarID	user id calendar id	Process the public calendar when having a remote access

Name	Service URL endpoint	Parameters	Expected Values	Description
	{calendarId}/{type}	type	INVALID_TYPE = -1 / PRIVATE_TYPE = 0 / SHARED_TYPE = 1 / PUBLIC_TYPE = 2	request. User must enter username and password to access.
upcomingEvent	/private/cs/calendar/getissues/{currentdatetime}/{type}/{limit}	currentdatatime type limit	valid time format INVALID_TYPE = -1 / PRIVATE_TYPE = 0 / SHARED_TYPE = 1 / PUBLIC_TYPE = 2 integer	The list of upcoming events in a specific calendar.
updateStatus	/private/cs/calendar/updatestatus/{taskid}	taskid	task id	Update the status of a task.
getCalendars	/private/cs/calendar/getcalendars	`no param`		Get a list of calendars.

Mail application

The Mail application of eXo Collaboration uses MailWebservice to provide all APIs for working with mail, such as sending/checking/storing mail to JCR.

REST API of the Mail portlet is exposed by *org.exoplatform.services.cs.mail.MailWebservice* class.

Service name	Service URL	Location	Description
MailWebservice	`$portalname/$restcontextname/private/cs/mail`	- Maven groupId: org.exoplatform.cs	This service is used to call extended services of the Mail application.

Service name	Service URL	Location	Description
		- artifactId: exo.cs.web.webservice	

- **APIs Usage**:

Use the following APIs to build all functions of the Mail application:

Name	Service URL endpoint	Parameters	Expected Values	Description
checkMail	/checkmail/{username}/{accountId}/{folderId/}/	username accountID folderId	user id account id folder id	Check mails when having a request.
synchFolders	/synchfolders/{username}/{accountId}/	username accountID	user id account id	Synchronize the mail folders in the clients and those in the mail server.
stopCheckMail	/stopcheckmail/{username}/{accountId}/	username accountId	user id account id	Stop checking the mail.
getCheckMailJobInfo	checkmailjobinfo/{username}/{accountId}/	username accountId	user id account id	The method to get information of the mail-checking job.
searchemail	/searchemail/{keywords}	keywords	string	Search information from emails.

Chat application

The Chat application uses some APIs to help users create a room, join a room, invite other users to room, or send files, and more.

The Chat application of eXo Collaboration uses two services: RESTXMPPService and FileExchangeService to do these tasks.

RESTXMPPService

REST API *RESTXMPPService* of the Chat portlet is exposed by *org.exoplatform.services.xmpp.rest.RESTXMPPService* class.

Service name	Service URL	Location	Description
RESTXMPPService	`$portalname/ $restcontextname/ xmpp`	* Maven groupid: org.exoplatform.cs * exo.cs.eXoApplication.chat.service	Implement all actions sent to the Chat server;

- **APIs Usage**:

Use the following APIs to build all functions of the Chat application:

Name	Service URL endpoint	Parameters	Expected Values	Description
loadJsResourceBundle	loadJsResourceBundle/ {locale}/	locale	locale id	Read the language files in the Chat server.
createRoom	/muc/ createroom/ {username}/	username room nickname	user id room name display name	Create a chat room or a group chat.
configRoom	/muc/configroom/ {username}/	username room	user id room name	Establish the configuration of a chat room.
getRoomConfigForm	/muc/ getroomconfig/ {username}/	username room	user id room name	Get the configuration of a chat room created.
getRoomInfo	/muc/ getroominfo/ {username}/	username room	user id string	Get the inforamtion of a chat room created.
getJoinedRooms	/muc/ joinedrooms/ {username}/	username	user id	List chat rooms that a user has been joined.
getRooms	/muc/rooms/ {username}/	username	user id	Get a list of group chat or chat rooms created.
declineToRoom	/muc/decline/ {username}/ {inviter}/	username inviter	user id user id	Refuse the invitation to join the chat room.

Name	Service URL endpoint	Parameters	Expected Values	Description
		room reason	room name string	
destroyRoom	/muc/destroy/{username}	username room reason altroom	user id room name string room id	Delete a chat room created.
inviteToRoom	/muc/invite/{username}/{invitee}/	username invitee room reason	user id user id room name string	Invite other users to join a chat room.
joinRoom	/muc/join/{username}/	username room nickname password	user id room name display name room password	Join a chat room.
leftRoom	/muc/leaveroom/{username}/	username room	user id room name	Leave a chat room.
changeNickname	/muc/changenickname/{username}/{nickname}/	username nickname	user id display name	Change the nickname of users.
changeAvailabilityStatusInRoom	/muc/changestatus/{username}/{mode}/	username mood	user id presence type	Change the status of a user in the chat room.

Name	Service URL endpoint	Parameters	Expected Values	Description
		room status	room name presence type	
changeSubject	muc/ changesubject/ {username}/	username room subject	user id room name string	Change the subject of a chat room.
manageRoleRoom	muc/ managerole/ {username}/	username room nickname role command	user id room name display name Participant / moderator grant/revoke	Change the role of each user in a chat room.
manageAffilationRoom	muc/ manageaffiliation/ {username}/	username room nickname affiliation command	user id room name display name String affiliation grant / revoke	Change the ownership of a chat room.
kickUserFromRoom	muc/kick/ {username}/	username nickname room reason	user id display name room name string	Remove a user from the chat room.

Name	Service URL endpoint	Parameters	Expected Values	Description
banUserFromRoom	/muc/ban/{username}/	username room name reason	user id room name user id string	Ban a user in the chat room.
addBoddyToRoster	/roster/add/{username}/{adduser}	username adduser nickname group	user id use id display name group id	Add a user into the contact list.
updateBoddy	/roster/update/{username}/{upduser}/	username upduser nickname group	user id user id display name group id	Update new users into the contact list.
createGroup	/roster/group/{username}/{group}/	username group	user id group id	Create a chat room.
askForSubscription	askforsubscription/{username}/{askuser}/	username askuser nickname	user id user id display name	Change the presence type of a user into Subscription.
cleanBuddylist	/rosterclean/{username}/	username	user id	Remove a user from the contact list.
getAllHistory	/history/getmessages/{usernameto}/{isGroupChat}/	usernameto isGroupChat	user id true / false	Get all the chat history of two users.

Name	Service URL endpoint	Parameters	Expected Values	Description
		usernamefrom	user id	
getHistoryBetweenDate	/history/getmessages/{usernameto}/{isGroupChat}/{from}/{to}/	usernameto isgroupchat from to usernamefrom	user id true / false valid date format valid date format user id	Get the chat history of two users in a specific period.
getHistoryFromDateToNow	/history/getmessages/{usernameto}/{isGroupChat}/{from}/	username isGroupChat from usernamefrom	user id true / false valid date format valid date format user id	Get the chat history of two users from a specific date to the current time.
getAllHistoryFile	/history/file/getmessages/{usernameto}/{isGroupChat}/{clientTimezoneOffset}/	uernameto isGroupChat clientTimezoneOffset usernamefrom	user id true / false Long user id	Download all the chat history file of two users.
getHistoryFromDateToNowFile	/history/file/getmessages/{usernameto}/{isGroupChat}/{from}/{clientTimezoneOffset}/	usernameto isGroupChat from clientTimezoneOffset usernamefrom	user id true / false valid date format Long user id	Download the chat history file of two users from a specific date to the current time

Name	Service URL endpoint	Parameters	Expected Values	Description
getHistoryBetweenDateFile	/getmessages/{usernameto}/{isGroupChat}/{from}/{to}/{clientTimezoneOffset}/	usernameto isGroupChat from to clientTimezoneOffset usernamefrom	user id true / false valid date format valid date format long user id	Download the chat history file of two users in the specific date.
getUserInfo	/getuserinfo/{username}/{needinfo}/	username needinfo	user id string	Get the information of a user.
login2	/login2/{forcache}/	forcache		Allow a user to log in the chat server.
logout	/logout/{username}/{presencestatus}/	username presencestatus	user id presencestatus	Allow a user to log out the chat server.
messageReceive	/history/messagereceive/{username}/{messageid}/	username messageid	user id message id	Receive a message from other users.
removeBuddy	/roster/del/{username}/{removeboddy}/	username removeboddy	user id user id	Delete a contact from the contact list.
removeTransport	/removetransport/{username}/{transport}/	username transport	user id transport servive (e.g: Yahoo, XMPP)	Reset the presence type at the service that is being used.
searchUsers	/searchuser/{username}/	username search	user id string	Search users in the chat server.

Name	Service URL endpoint	Parameters	Expected Values	Description
		byUsername byName byEmail searchService	true / false true /false true / false string	
sendMessage	/sendmessage/{username}/	username messageBean	usesr id object	Send an message to other users.
sendMUCMessage	/muc/sendmessage/{username}/	username messageBean	user id object	Send a message to multile users or a group.
setUserStatus	/sendstatus/{username}/{status}/	username status	user id available/ unavailabe / do not disturb / away / extend away	Change the status of a user.
subscribeUser	/subscribeuser/{username}/{subsuser}/	username subsuser	user id user id	Change presence type into Subcribed type.
unsubscribeUser	/unsubscribeuser/{username}/{unsubsuser}/	username unsubsuser	user id user id	Change presence type into the Unsubscribed type.
acceptFile	/fileexchange/accept/{username}/{uuid}/	username uuid	user id string	Accept getting a file sent from another user.
rejectFile	/fileexchange/reject/{username}/{uuid}/	username uuid	user id string	Refuse getting a file sent from another user.

Name	Service URL endpoint	Parameters	Expected Values	Description
getPreviousStatus	/getprevstatus/{username}/	username	user id	Get the tatus of a user in the last log-in.

FileExchangeService

REST API *FileExchangeService* for uploading files is defined in *org.exoplatform.services.xmpp.rest.FileExchangeService* class.

Service name	Service URL	Location	Description
FileExchangeService	`$portalname/$restcontextname/fileexchange`	- Maven groupid: org.exoplatform.cs - InterfactId: exo.cs.eXoApplication.chat.service	Upload a file to the server and inform the user that the file can be downloaded to the local computer.

- APIs usage: Use the following APIs to upload and send files to other users:

Name	Service URL endpoint	Parameters	Expected Values	Description
upload	$portalname/$restcontextname/fileexchange	description username requestor isroom	string user id user id true / false	Upload a file to the server.

eXo Platform 3.0 - Knowledge Functions

Prerequisites

- *http://wiki.exoplatform.org/xwiki/bin/view/Portal/Portal%20Manual#HDevelopment* [Portlets & Gadgets]
- *http://wiki.exoplatform.org/xwiki/bin/view/WS/* [REST APIs]

Applications

Portlets

Forum Portlet

Overview

Portlet name	War name	Description
ForumPortlet	forum.war	The Forum portlet is the application for users to post and read messages on different topics.

- Declaration template

```
<portlet-application>
  <portlet>
    <application-ref>forum</application-ref>
    <portlet-ref>ForumPortlet</portlet-ref>
  </portlet>
</portlet-application>
```

Portlet.xml

- See the *portlet.xml* file in the project following this path: *forum/WEB-INF/portlet.xml*.

Preferences

Preference name	Possible value	Default value	Description
useAjax	`true`, `false`	`true`	Define if links in the Forum will be plain hrefs or javascript ajax (better for SEO) or not.
showForumActionBar	`true`, `false`	`true`	This is the UIForumActionBar. If the value is set to "true", the UIForumActionBar will be shown. If false, the

Preference name	Possible value	Default value	Description
			UIForumActionBar will be hidden.
forumNewPost	`day number`	`1`	Specify if a post is new. If the post is created within the set period, it is new in the Forum.
enableIPLogging	`true`, `false`	`true`	Enable the IP logging function in the Forum. IP addresses of all posts will be collected.
enableIPFiltering	`true`, `false`	`true`	Enable the IP filter function in Forum, enabling IP addresses to be blocked in the Forum.
invisibleCategories	`id categories`	`empty`	Hide some categories. If the value is set empty, all categories of the Forum will be shown.
invisibleForums	`id forums`	`empty`	Hide some Forums. All Forums will be shown if the value is set empty.
uploadFileSizeLimitMB	`integer`	`20`	Limit the size of uploaded files in MB in the Forum.
isShowForumJump	`true`, `false`	`true`	Specify if the Forum jump panel is shown or not.
isShowIconsLegend	`true`, `false`	`true`	Specify if the icon legends panel is shown or not.
isShowModerators	`true`, `false`	`true`	Specify if the moderators panel is shown or not.
isShowPoll	`true`, `false`	`true`	Specify if the poll panel is shown or not.

Preference name	Possible value	Default value	Description
isShowQuickReply	`true`, `false`	`true`	Specify if the quick reply panel is shown or not.
isShowRules	`true`, `false`	`true`	Specify if the forum rules panel is shown or not.
isShowStatistics	`true`, `false`	`true`	Specify if the statistics panel is shown or not.

Events

Name	Description
ForumLinkEvent	Set the render for UIForumLinkPortlet and set values for UIForumLinks.
ReLoadPortletEvent	Reload UIForumPortlet.
OpenLink	Update values for UIForumLinks.
ForumPollEvent	Set the render for UIForumPollPortlet.
ForumModerateEvent	Set the render for UIForumModeratorPortlet.
ForumRuleEvent	Set the render for UIForumRulePortlet.
QuickReplyEvent	Set the render for UIForumQuickReplyPortlet.

ForumLinkEvent

This event is fired through UIForumLinkPortlet.

To receive ForumLinkEvent, you must use the *ForumParameter* class with two properties:

Name	Type	Possible value	Description
isRenderForumLink	`boolean`	true/false	If the value is set to true or false, the Forum link is rendered or not respectively.
path	`string`	The absolute path of the class node (including types: category, forum, topic) defined by JCR.	Set data for the UIForumLinkPortlet.

For example:

```
PortletRequestContext pcontext = (PortletRequestContext)
 WebuiRequestContext.getCurrentInstance();
ActionResponse actionRes = (ActionResponse) pcontext.getResponse();
ForumParameter param = new ForumParameter();
String path = forum.getPath();
if ( ...condition to render the UIForumLinkPortlet... ) {
  param.setRenderForumLink(true);
  param.setPath(path);
} else {
  param.setRenderForumLink(false);
}
actionRes.setEvent(new QName("ForumLinkEvent"), param) ;
```

ReLoadPortletEvent

This event is fired through UIForumPortlet.

To receive ReLoadPortletEvent, you must use the *ForumParameter* class with two properties:

Name	Type	Possible value	Description
topicId	`string`	Id of topic.	Return the id of topic for UIForumPortlet
isRenderPoll	`boolean`	true/false	If the value is set to true or false, the *UITopicPoll* component is rendered or not respectively.

For example:

```
....
ActionResponse actionRes = pcontext.getResponse() ;
ForumParameter param = new ForumParameter() ;
param.setRenderPoll(true);
param.setTopicId(topic.get());
actionRes.setEvent(new QName("ReLoadPortletEvent"), param) ;
....
```

OpenLink

This event is fired through UIForumPortlet.

To receive OpenLink, you must use the *ForumParameter* class with one property:

Name	Type	Possible value	Description
path	`string`	The absolute path of the node defined by JCR.	Set data for the UIForumPortlet.

For example:

```
....
ActionResponse actionRes = pcontext.getResponse() ;
ForumParameter param = new ForumParameter() ;
param.setPath(path);
actionRes.setEvent(new QName("OpenLink"), param) ;
....
```

ForumPollEvent

This event is fired through UIForumPollPortlet.

To receive ForumPollEvent, you must use the *ForumParameter* class with four properties:

Name	Type	Possible value	Description
isRenderPoll	`boolean`	True/false	If the value is set to true or false, the *UIForumPollPortlet* portlet is rendered or not respectively.
categoryId	`string`	Id of category	Return the Id of category for UIForumPollPortlet.
forumId	`string`	Id of forum	Return the Id of forum for UIForumPollPortlet.
topicId	`string`	Id of topic	Return the Id of topic for UIForumPollPortlet.

For example:

```
```

```
....
ActionResponse actionRes = pcontext.getResponse() ;
ForumParameter param = new ForumParameter() ;
param.setCategoryId(categoryId) ;
param.setForumId(forumId);
param.setTopicId(topicId);
param.setRenderPoll(topic.getIsPoll());
actionRes.setEvent(new QName("ForumPollEvent"), param);
....
```

ForumModerateEvent

This event is fired through UIForumModeratePortlet.

To receive ForumModerateEvent, you must use the *ForumParameter* class with two properties:

Name	Type	Possible value	Description
isRenderModerator	`boolean`	True/false	If the value is set to true or false, the *UIForumModeratePortlet* portlet is rendered or not respectively.
moderator	`list of strings`	List of user name	Set data for UIForumModeratePortlet.

For example:

```
....
List<String> moderators = Arays.asList(forum.getModerators());
ActionResponse actionRes = pcontext.getResponse() ;
ForumParameter param = new ForumParameter() ;
param.setModerators(moderators);
param.setRenderModerator(true);
actionRes.setEvent(new QName("ForumPollEvent"), param);
....
```

ForumRuleEvent

This event is fired through UIForumRulePortlet.

To receive ForumRuleEvent, you must use the *ForumParameter* class with two properties:

Name	Type	Possible value	Description
isRenderRule	`boolean`	True/false	If the value is set to true or false, the *UIForumRulePortlet* portlet is rendered or not respectively.
infoRules	`list of strings`	The list of states: **can create topic**, **can add post** and **topic has lock**.	Set permissions for users in UIForumRulePortlet.

For example:

```
....
ActionResponse actionRes = pcontext.getResponse() ;
ForumParameter param = new ForumParameter() ;
List<String> list = param.getInfoRules();
if(forum.getIsClosed() || forum.getIsLock()) {
  list.set(0, "true");
} else {
  list.set(0, "false");
}
list.set(1, String.valueOf(canCreateTopic));
list.set(2, String.valueOf(isCanPost));
param.setInfoRules(list);
param.setRenderRule(true);
actionRes.setEvent(new QName("ForumRuleEvent"), param) ;
....
```

QuickReplyEvent

This event is fired through UIQuickReplyPortlet.

To receive QuickReplyEvent, you must use the *ForumParameter* class with five properties:

Name	Type	Possible value	Description
isRenderQuickReply	`boolean`	True/false	If the value is set to true or false, the *UIQuickReplyPortlet*

Name	Type	Possible value	Description
			portlet is rendered or not respectively.
isModerator	`boolean`	True/false	Specify if the user is moderator of forum containing the topic with quick reply or not.
categoryId	`string`	Id of category	Return the Id of category for UIQuickReplyPortlet.
forumId	`string`	Id of forum	Return the Id of forum for UIQuickReplyPortlet.
topicId	`string`	Id of topic	Return the Id of topic for UIQuickReplyPortlet.

For example:

```
....
ActionResponse actionRes = pcontext.getResponse() ;
ForumParameter param = new ForumParameter() ;
param.setRenderQuickReply(isCanPost);
param.setModerator(isMod);
param.setCategoryId(categoryId) ;
param.setForumId(forumId);
param.setTopicId(topicId);
actionRes.setEvent(new QName("QuickReplyEvent"), param) ;;
....
```

Answers Portlet

Overview

Portlet name	War name	Description
AnswersPortlet	faq.war	The Answers portlet is the application to create answers, to reply and manage questions.

Portlet.xml

- See the *portlet.xml* file in the project following this path: */webapps/faq/WEB-INF/portlet.xml*.

Portlet Preferences

The Answers portlet consists of preferences as follows:

Preference name	Possible value	Default value	Description
enableViewAvatar	`true`, `false`	`true`	Enable users to view the avatar of owner posting the question.
enableAutomaticRSS	`true`, `false`	`true`	Enable users to get RSS automatically.
enableVotes AndComments	`true`, `false`	`true`	Enable users to give votes and comments for the question.
enableAnonymous SubmitQuestion	`true`, `false`	`true`	Enable anonymous users to submit questions.
display	`approved`, `both`	`both`	Enable administrators to view unapproved questions in the questions list in UIQuestions.
SendMailAdd NewQuestion	`string`	`empty`	Display the content of sent email when a new question is added.
SendMailEdit ResponseQuestion	`string`	`empty`	Display the email content when a response is edited.
emailMoveQuestion	`string`	`empty`	Display the email content when a question is moved.
orderBy	`alphabet`, `created`	`alphabet`	Arrange questions in the alphabet or created date order.
orderType	`asc`, `desc`	`asc`	Display questions in the ascending or descending order.
isDiscussForum	`true`, `false`	`false`	

Preference name	Possible value	Default value	Description
			Enable the DiscussQuestions function.
idNameCategoryForum	`CategoryName,ForumName`	`empty`	Select categories and forums for the DiscussionQuestions function.
uploadFileSizeLimitMB	`Integer`	`20`	Set the maximum size of uploaded files in MB.

FAQ Portlet

Overview

Portlet name	War name	Description
FAQPortlet	faq.war	The FAQ portlet which is the application to show questions and answers.

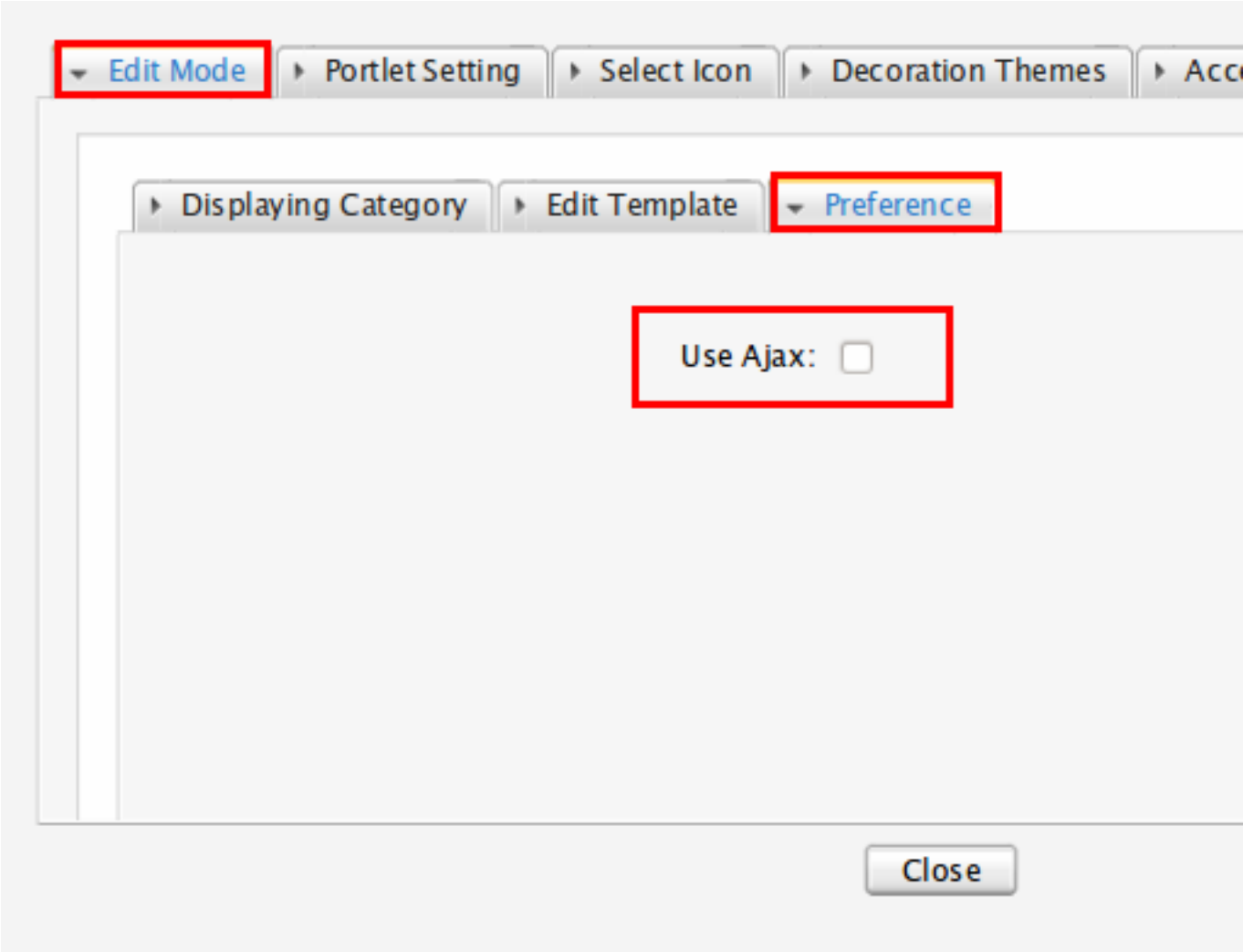
Portlet.xml

- See the *portlet.xml* file in the project following this path: */webapps/faq/WEB-INF/portlet.xml*.

Portlet Preferences

The FAQ portlet includes some portlet preferences that you can configure to alter the behavior.

At runtime, you can use the EDIT mode portlet to set the preferences:



Alternatively, you can configure the portlet in the *portet-preferences.xml* file.

Preference name	Possible value	Default value	Description
useAjax	`true`, `false`	`false`	Specify if the AJAX load or HREF load is used.

Polls Portlet

Overview

Portlet name	War name	Description
PollPortlet	poll.war	The Poll portlet which is the application for users to vote any ideas, or activities.

Portlet.xml

- See the *portlet.xml* file in the project following this path: *poll/WEB-INF/portlet.xml*.

```
<portlet-preferences>
  <preference>
    <name>pollIdShow</name>
    <value/> <!-- PollId -->
    <read-only>false</read-only>
  </preference>
</portlet-preferences>
```

Portlet Preferences

Preference name	Possible value	Default value	Description
pollIdShow	`string`	`empty`	The Id of poll which is displayed in the Polls portlet.

Gadgets

Overview

eXo Platform provides a gadget which enables users to see a poll. The Poll Gadget is developed on the combination of Gagget by GateIn and Polls Service. The Poll Gadget allows users to apply functions of Polls, such as viewing and voting Polls.

Gadget name	War name	Description
pollslist	poll.war	The list of Polls.

Preferences

Preference name	Description
pollId	The Id of Polls which is displayed in the Polls gadget.

Links to used REST services

- portal/rest/private/ks/poll/viewpoll/pollId

- portal/rest/private/ks/poll/votepoll/pollId/indexVote

Configuration

Components

Components of eXo Knowledge

Key	Data type	Description
org.exoplatform.ks.common.jcr.KSDataLocation	org.exoplatform.ks.common.jcr.KSDataLocation	Hold the JCR storage location for eXo Knowledge data.
org.exoplatform.services.scheduler.JobSchedulerService	org.exoplatform.services.scheduler.impl.JobSchedulerServiceImpl	Define a job to execute a given number of times during a given period. It is used to monitor jobs automatically and continously, to schedule event-driven jobs and reports, and to control performance.

- Init-params of org.exoplatform.ks.common.jcr.KSDataLocation:

Name	Possible value	Default value
repository	`string`	`repository`
workspace	`string`	`knowledge`

Components of Forum

Key	Data type	Description
org.exoplatform.ks.bbcode.api.BBCodeService	org.exoplatform.ks.bbcode.core.BBCodeServiceImpl	Manage CRUD operations on BBCodes.
org.exoplatform.forum.service.DataStorage	org.exoplatform.forum.service.impl.JCRDataStorage	Store data of Forum via the JCR system.
org.exoplatform.forum.service.ForumService	org.exoplatform.forum.service.impl.ForumServiceImpl	Include all public APIs to interact with the UI component and database.
org.exoplatform.forum.service.ForumStatisticsService	org.exoplatform.forum.service.impl.ForumStatisticsServiceImpl	Include all public APIs to interact with the database of Statistics system.

Components of Answers

Key	Data type	Description
org.exoplatform.faq.service.FAQService	org.exoplatform.faq.service.impl.FAQServiceImpl	

Key	Data type	Description
		Include all public APIs to interact with the UI component and database.
org.exoplatform.faq.service.DataStorage	org.exoplatform.faq.service.impl.JCRDataStorage	Store data of FAQ via the JCR system.

Components of Polls

Key	Data type	Description
org.exoplatform.poll.service.DataStorage	org.exoplatform.poll.service.impl.JCRDataStorage	Include all public APIs to interact with the UI component and database.
org.exoplatform.poll.service.PollService	org.exoplatform.poll.service.impl.PollServiceImpl	Store data of Polls via the JCR system.

External-component-plugin

Init data configuration

Init data

Application	Component	Description
Forum	`KSDataLocation, ForumServiceImpl`	Initialize default data of the Forum portlet.
Answers	`KSDataLocation, AnswerServiceImpl`	Initialize default data of the Answers portlet.
Polls	`KSDataLocation, PollServiceImpl`	Initialize default data of the Polls portlet.

The Init data plug-in is used to define the default data in the *.xml* file. It includes nodes (node of jcr).

When the *org.exoplatform.services.jcr.config.RepositoryServiceConfiguration* component is initialized, the *org.exoplatform.services.jcr.impl.config.RepositoryServiceConfigurationPlugin* component will be get and the function *addConfig* is called. Then, the */ks-extension/jcr/repository-configuration.xml* file is loaded and the component *org.exoplatform.ks.common.jcr.KSDataLocation* will be initialized. Next, the *setLocation* function is called, setting up the workspace and repository for KS. After that, the *addPlugin* function will be run, generating the DataLocation (some parent nodes) for eXo Knowledge.

Configuration

Initialize the conf-part for loading *repository-configuration.xml*

When the server starts, the *jcr-configuration.xml* file is initialized. The component-plugin named addConfig will be referred to org.exoplatform.services.jcr.impl.config.RepositoryServiceConfigurationPlugin to load the *war:/ks-extension/jcr/repository-configuration.xml_ file.*

```
<component-plugin>
  <!-- The name of the plugin -->
  <name>Sample RepositoryServiceConfiguration Plugin</name>
  <!-- The name of the method to call on the RepositoryServiceConfiguration
    in order to add the RepositoryServiceConfigurations -->
  <set-method>addConfig</set-method>
  <!-- The full qualified name of the RepositoryServiceConfigurationPlugin -->
  <type>org.exoplatform.services.jcr.impl.config.RepositoryServiceConfigurationPlugin</type>
  <init-params>
    <value-param>
      <name>conf-path</name>
      <description>JCR configuration file</description>
      <value>war:/ks-extension/jcr/repository-configuration.xml</value>
    </value-param>
  </init-params>
</component-plugin>
```

- In which:

Name	Set-method	Type	Description
RepositoryService ConfigurationPlugin	addConfig	org.exoplatform.services.jcr.impl.config.RepositoryServiceConfigurationPlugin	Read the configuration of JCR data to initialize data.

- Init-params

Name	Possible value	Default value	Description
conf-path	`string`	war:/ks-extension/jcr/repository-configuration.xml	The path to the *repository-configuration.xml* file.

Initialize workspace name and repository name in *storage-configuration.xml*

In details:

Once the *war:/ks-extension/jcr/repository-configuration.xml* file has been initialized, the server will load the *storage-configuration.xml* file, and the *setLocation* function in the *org.exoplatform.ks.common.conf.DataLocationPlugin* component will run.

```
<external-component-plugins>
  <target-component>org.exoplatform.ks.common.jcr.KSDataLocation</target-component>
  <component-plugin>
    <name>ks.data.location</name>
    <set-method>setLocation</set-method>
    <type>org.exoplatform.ks.common.conf.DataLocationPlugin</type>
    <init-params>
      <value-param>
        <name>repository</name>
        <description>JCR repository for KS data</description>
        <value>repository</value>
      </value-param>
      <value-param>
        <name>workspace</name>
        <description>workspace for KS data</description>
        <value>knowledge</value>
      </value-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

In which,

Value-param	Possible value	Default value	Description
repository	`string`	`repository`	The JCR repository for KS data.
workspace	`string`	`knowledge`	The workspace for KS data.

Initialize data

Once the workspace name and repository name are registered, the server will load org.exoplatform.services.jcr.ext.hierarchy.NodeHierarchyCreator and the *addPaths* function in org.exoplatform.services.jcr.ext.hierarchy.impl.AddPathPlugin is called. Then, the data location will be built.

```
<component-plugin>
```

```
<name>addPaths</name>
<set-method>addPlugin</set-method>
<type>org.exoplatform.services.jcr.ext.hierarchy.impl.AddPathPlugin</type>
<init-params>
  <object-param>
    <name>ks.storage</name>
    <description>ks data storage tree</description>
    <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig">
      <field name="repository">
        <string>repository</string>
      </field>
      <field name="workspaces">
        <collection type="java.util.ArrayList">
          <value>
            <string>knowledge</string>
          </value>
        </collection>
      </field>
      <field name="jcrPaths">
        <collection type="java.util.ArrayList">
          <value>
           <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
              <field name="alias">
                <string>eXoApplications</string>
              </field>
              <field name="path">
                <string>/exo:applications</string>
              </field>
              <field name="permissions">
                <collection type="java.util.ArrayList">
                  <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                      <field name="identity">
                        <string>*:/platform/administrators</string>
                      </field>
                      <field name="read">
                        <string>true</string>
                      </field>
                      <field name="addNode">
                        <string>true</string>
                      </field>
                      <field name="setProperty">
                        <string>true</string>
```

```
                </field>
                <field name="remove">
                  <string>true</string>
                </field>
              </object>
            </value>
          </collection>
        </field>
      </object>
    </value>

    <value>
     <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
        <field name="alias">
          <string>ksUserAvatar</string>
        </field>
        <field name="path">
          <string>/exo:applications/ksUserAvatar</string>
        </field>
        <field name="nodeType">
          <string>nt:unstructured</string>
        </field>
        <field name="permissions">
          <collection type="java.util.ArrayList">
            <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                <field name="identity">
                  <string>*:/platform/administrators</string>
                </field>
                <field name="read">
                  <string>true</string>
                </field>
                <field name="addNode">
                  <string>true</string>
                </field>
                <field name="setProperty">
                  <string>true</string>
                </field>
                <field name="remove">
                  <string>true</string>
                </field>
              </object>
            </value>
```

```
            </collection>
          </field>
        </object>
      </value>

      <value>
       <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
          <field name="alias">
            <string>ForumService</string>
          </field>
          <field name="path">
            <string>/exo:applications/ForumService</string>
          </field>
          <field name="nodeType">
            <string>exo:forumHome</string>
          </field>
          <field name="permissions">
            <collection type="java.util.ArrayList">
              <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                  <field name="identity">
                    <string>*:/platform/administrators</string>
                  </field>
                  <field name="read">
                    <string>true</string>
                  </field>
                  <field name="addNode">
                    <string>true</string>
                  </field>
                  <field name="setProperty">
                    <string>true</string>
                  </field>
                  <field name="remove">
                    <string>true</string>
                  </field>
                </object>
              </value>
            </collection>
          </field>
        </object>
      </value>

      <value>
```

```
        <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
          <field name="alias">
            <string>ForumSystem</string>
          </field>
          <field name="path">
            <string>/exo:applications/ForumService/ForumSystem</string>
          </field>
          <field name="nodeType">
            <string>exo:forumSystem</string>
          </field>
          <field name="permissions">
            <collection type="java.util.ArrayList">
              <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                  <field name="identity">
                    <string>*:/platform/administrators</string>
                  </field>
                  <field name="read">
                    <string>true</string>
                  </field>
                  <field name="addNode">
                    <string>true</string>
                  </field>
                  <field name="setProperty">
                    <string>true</string>
                  </field>
                  <field name="remove">
                    <string>true</string>
                  </field>
                </object>
              </value>
            </collection>
          </field>
        </object>
      </value>

      <value>
       <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
          <field name="alias">
            <string>UserProfileHome</string>
          </field>
          <field name="path">
            <string>/exo:applications/ForumService/ForumSystem/UserProfileHome</string>
```

```
            </field>
            <field name="nodeType">
              <string>exo:userProfileHome</string>
            </field>
            <field name="permissions">
              <collection type="java.util.ArrayList">
                <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                    <field name="identity">
                      <string>*:/platform/administrators</string>
                    </field>
                    <field name="read">
                      <string>true</string>
                    </field>
                    <field name="addNode">
                      <string>true</string>
                    </field>
                    <field name="setProperty">
                      <string>true</string>
                    </field>
                    <field name="remove">
                      <string>true</string>
                    </field>
                  </object>
                </value>
              </collection>
            </field>
          </object>
        </value>

        <value>
         <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
            <field name="alias">
              <string>StatisticHome</string>
            </field>
            <field name="path">
              <string>/exo:applications/ForumService/ForumSystem/StatisticHome</string>
            </field>
            <field name="nodeType">
              <string>exo:statisticHome</string>
            </field>
            <field name="permissions">
              <collection type="java.util.ArrayList">
```

```
              <value>
                                                                                <object
 type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                  <field name="identity">
                    <string>*:/platform/administrators</string>
                  </field>
                  <field name="read">
                    <string>true</string>
                  </field>
                  <field name="addNode">
                    <string>true</string>
                  </field>
                  <field name="setProperty">
                    <string>true</string>
                  </field>
                  <field name="remove">
                    <string>true</string>
                  </field>
                </object>
              </value>
            </collection>
          </field>
        </object>
      </value>

      <value>
       <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
          <field name="alias">
            <string>forumStatistic</string>
          </field>
          <field name="path">
           <string>/exo:applications/ForumService/ForumSystem/StatisticHome/forumStatistic</
string>
          </field>
          <field name="nodeType">
            <string>exo:forumStatistic</string>
          </field>
          <field name="permissions">
            <collection type="java.util.ArrayList">
              <value>
                                                                                <object
 type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                  <field name="identity">
                    <string>*:/platform/administrators</string>
```

```
                </field>
                <field name="read">
                  <string>true</string>
                </field>
                <field name="addNode">
                  <string>true</string>
                </field>
                <field name="setProperty">
                  <string>true</string>
                </field>
                <field name="remove">
                  <string>true</string>
                </field>
              </object>
            </value>
          </collection>
        </field>
      </object>
    </value>

    <value>
     <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
        <field name="alias">
          <string>AdministrationHome</string>
        </field>
        <field name="path">
          <string>/exo:applications/ForumService/ForumSystem/AdministrationHome</string>
        </field>
        <field name="nodeType">
          <string>exo:administrationHome</string>
        </field>
        <field name="permissions">
          <collection type="java.util.ArrayList">
            <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                <field name="identity">
                  <string>*:/platform/administrators</string>
                </field>
                <field name="read">
                  <string>true</string>
                </field>
                <field name="addNode">
                  <string>true</string>
```

```
                </field>
                <field name="setProperty">
                  <string>true</string>
                </field>
                <field name="remove">
                  <string>true</string>
                </field>
              </object>
            </value>
          </collection>
        </field>
      </object>
    </value>

    <value>
     <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
        <field name="alias">
          <string>BanIPHome</string>
        </field>
        <field name="path">
          <string>/exo:applications/ForumService/ForumSystem/BanIPHome</string>
        </field>
        <field name="nodeType">
          <string>exo:banIPHome</string>
        </field>
        <field name="permissions">
          <collection type="java.util.ArrayList">
            <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                <field name="identity">
                  <string>*:/platform/administrators</string>
                </field>
                <field name="read">
                  <string>true</string>
                </field>
                <field name="addNode">
                  <string>true</string>
                </field>
                <field name="setProperty">
                  <string>true</string>
                </field>
                <field name="remove">
                  <string>true</string>
```

```
                </field>
               </object>
              </value>
             </collection>
            </field>
           </object>
          </value>

          <value>
           <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
            <field name="alias">
             <string>forumBanIP</string>
            </field>
            <field name="path">
          <string>/exo:applications/ForumService/ForumSystem/BanIPHome/forumBanIP</string>
            </field>
            <field name="nodeType">
             <string>exo:banIP</string>
            </field>
            <field name="permissions">
             <collection type="java.util.ArrayList">
              <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                <field name="identity">
                 <string>*:/platform/administrators</string>
                </field>
                <field name="read">
                 <string>true</string>
                </field>
                <field name="addNode">
                 <string>true</string>
                </field>
                <field name="setProperty">
                 <string>true</string>
                </field>
                <field name="remove">
                 <string>true</string>
                </field>
               </object>
              </value>
             </collection>
            </field>
           </object>
```

```
        </value>

        <value>
         <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
           <field name="alias">
             <string>ForumData</string>
           </field>
           <field name="path">
             <string>/exo:applications/ForumService/ForumData</string>
           </field>
           <field name="nodeType">
             <string>exo:forumData</string>
           </field>
           <field name="permissions">
             <collection type="java.util.ArrayList">
               <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                 <field name="identity">
                   <string>*:/platform/administrators</string>
                 </field>
                 <field name="read">
                   <string>true</string>
                 </field>
                 <field name="addNode">
                   <string>true</string>
                 </field>
                 <field name="setProperty">
                   <string>true</string>
                 </field>
                 <field name="remove">
                   <string>true</string>
                 </field>
               </object>
             </value>
           </collection>
         </field>
        </object>
        </value>

        <value>
         <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
           <field name="alias">
             <string>TopicTypeHome</string>
```

```
            </field>
            <field name="path">
              <string>/exo:applications/ForumService/ForumData/TopicTypeHome</string>
            </field>
            <field name="nodeType">
              <string>exo:topicTypeHome</string>
            </field>
            <field name="permissions">
              <collection type="java.util.ArrayList">
                <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                    <field name="identity">
                      <string>*:/platform/administrators</string>
                    </field>
                    <field name="read">
                      <string>true</string>
                    </field>
                    <field name="addNode">
                      <string>true</string>
                    </field>
                    <field name="setProperty">
                      <string>true</string>
                    </field>
                    <field name="remove">
                      <string>true</string>
                    </field>
                  </object>
                </value>
              </collection>
            </field>
          </object>
        </value>

        <value>
         <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
            <field name="alias">
              <string>CategoryHome</string>
            </field>
            <field name="path">
              <string>/exo:applications/ForumService/ForumData/CategoryHome</string>
            </field>
            <field name="nodeType">
              <string>exo:categoryHome</string>
```

```
            </field>
            <field name="permissions">
              <collection type="java.util.ArrayList">
                <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                  <field name="identity">
                    <string>*:/platform/administrators</string>
                  </field>
                  <field name="read">
                    <string>true</string>
                  </field>
                  <field name="addNode">
                    <string>true</string>
                  </field>
                  <field name="setProperty">
                    <string>true</string>
                  </field>
                  <field name="remove">
                    <string>true</string>
                  </field>
                </object>
              </value>
            </collection>
          </field>
        </object>
      </value>

      <value>
       <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
          <field name="alias">
            <string>TagHome</string>
          </field>
          <field name="path">
            <string>/exo:applications/ForumService/ForumData/TagHome</string>
          </field>
          <field name="nodeType">
            <string>exo:tagHome</string>
          </field>
          <field name="permissions">
            <collection type="java.util.ArrayList">
              <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
```

```
                <field name="identity">
                  <string>*:/platform/administrators</string>
                </field>
                <field name="read">
                  <string>true</string>
                </field>
                <field name="addNode">
                  <string>true</string>
                </field>
                <field name="setProperty">
                  <string>true</string>
                </field>
                <field name="remove">
                  <string>true</string>
                </field>
              </object>
            </value>
          </collection>
        </field>
      </object>
    </value>

    <value>
     <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
        <field name="alias">
          <string>forumBBCode</string>
        </field>
        <field name="path">
          <string>/exo:applications/ForumService/ForumData/forumBBCode</string>
        </field>
        <field name="nodeType">
          <string>exo:forumBBCodeHome</string>
        </field>
        <field name="permissions">
          <collection type="java.util.ArrayList">
            <value>
                                                                        <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                <field name="identity">
                  <string>*:/platform/administrators</string>
                </field>
                <field name="read">
                  <string>true</string>
                </field>
```

```
                <field name="addNode">
                  <string>true</string>
                </field>
                <field name="setProperty">
                  <string>true</string>
                </field>
                <field name="remove">
                  <string>true</string>
                </field>
              </object>
            </value>
          </collection>
        </field>
      </object>
    </value>

    <value>
     <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
        <field name="alias">
          <string>faqApp</string>
        </field>
        <field name="path">
          <string>/exo:applications/faqApp</string>
        </field>
        <field name="nodeType">
          <string>exo:faqHome</string>
        </field>
        <field name="permissions">
          <collection type="java.util.ArrayList">
            <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                <field name="identity">
                  <string>*:/platform/administrators</string>
                </field>
                <field name="read">
                  <string>true</string>
                </field>
                <field name="addNode">
                  <string>true</string>
                </field>
                <field name="setProperty">
                  <string>true</string>
                </field>
```

```
                <field name="remove">
                  <string>true</string>
                </field>
              </object>
            </value>
          </collection>
        </field>
      </object>
    </value>

    <value>
     <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
        <field name="alias">
          <string>settingHome</string>
        </field>
        <field name="path">
          <string>/exo:applications/faqApp/settingHome</string>
        </field>
        <field name="nodeType">
          <string>exo:faqSettingHome</string>
        </field>
        <field name="permissions">
          <collection type="java.util.ArrayList">
            <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                <field name="identity">
                  <string>*:/platform/administrators</string>
                </field>
                <field name="read">
                  <string>true</string>
                </field>
                <field name="addNode">
                  <string>true</string>
                </field>
                <field name="setProperty">
                  <string>true</string>
                </field>
                <field name="remove">
                  <string>true</string>
                </field>
              </object>
            </value>
          </collection>
```

```
        </field>
       </object>
      </value>

      <value>
       <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
        <field name="alias">
         <string>userSettingHome</string>
        </field>
        <field name="path">
         <string>/exo:applications/faqApp/settingHome/userSettingHome</string>
        </field>
        <field name="nodeType">
         <string>exo:faqUserSettingHome</string>
        </field>
        <field name="permissions">
         <collection type="java.util.ArrayList">
          <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
             <field name="identity">
              <string>*:/platform/administrators</string>
             </field>
             <field name="read">
              <string>true</string>
             </field>
             <field name="addNode">
              <string>true</string>
             </field>
             <field name="setProperty">
              <string>true</string>
             </field>
             <field name="remove">
              <string>true</string>
             </field>
            </object>
           </value>
         </collection>
        </field>
       </object>
      </value>

      <value>
       <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
```

```
            <field name="alias">
              <string>categories</string>
            </field>
            <field name="path">
              <string>/exo:applications/faqApp/categories</string>
            </field>
            <field name="nodeType">
              <string>exo:faqCategory</string>
            </field>
            <field name="mixinTypes">
              <collection type="java.util.ArrayList">
                <value>
                  <string>mix:faqSubCategory</string>
                </value>
              </collection>
            </field>
            <field name="permissions">
              <collection type="java.util.ArrayList">
                <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                    <field name="identity">
                      <string>*:/platform/administrators</string>
                    </field>
                    <field name="read">
                      <string>true</string>
                    </field>
                    <field name="addNode">
                      <string>true</string>
                    </field>
                    <field name="setProperty">
                      <string>true</string>
                    </field>
                    <field name="remove">
                      <string>true</string>
                    </field>
                  </object>
                </value>
              </collection>
            </field>
          </object>
        </value>

        <value>
```

```
        <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
          <field name="alias">
            <string>templateHome</string>
          </field>
          <field name="path">
            <string>/exo:applications/faqApp/templateHome</string>
          </field>
          <field name="nodeType">
            <string>exo:templateHome</string>
          </field>
          <field name="permissions">
            <collection type="java.util.ArrayList">
              <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                  <field name="identity">
                    <string>*:/platform/administrators</string>
                  </field>
                  <field name="read">
                    <string>true</string>
                  </field>
                  <field name="addNode">
                    <string>true</string>
                  </field>
                  <field name="setProperty">
                    <string>true</string>
                  </field>
                  <field name="remove">
                    <string>true</string>
                  </field>
                </object>
              </value>
            </collection>
          </field>
        </object>
      </value>

      <value>
       <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
          <field name="alias">
            <string>eXoPolls</string>
          </field>
          <field name="path">
            <string>/exo:applications/eXoPolls</string>
```

```
          </field>
          <field name="nodeType">
            <string>nt:unstructured</string>
          </field>
          <field name="permissions">
            <collection type="java.util.ArrayList">
              <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                  <field name="identity">
                    <string>*:/platform/administrators</string>
                  </field>
                  <field name="read">
                    <string>true</string>
                  </field>
                  <field name="addNode">
                    <string>true</string>
                  </field>
                  <field name="setProperty">
                    <string>true</string>
                  </field>
                  <field name="remove">
                    <string>true</string>
                  </field>
                </object>
              </value>
            </collection>
          </field>
        </object>
      </value>

      <value>
       <object type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$JcrPath">
          <field name="alias">
            <string>groupsPath</string>
          </field>
          <field name="path">
            <string>/Groups</string>
          </field>
          <field name="permissions">
            <collection type="java.util.ArrayList">
              <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
```

```
                    <field name="identity">
                      <string>*:/platform/administrators</string>
                    </field>
                    <field name="read">
                      <string>true</string>
                    </field>
                    <field name="addNode">
                      <string>true</string>
                    </field>
                    <field name="setProperty">
                      <string>true</string>
                    </field>
                    <field name="remove">
                      <string>true</string>
                    </field>
                  </object>
                </value>
                <value>
                                                                                <object
type="org.exoplatform.services.jcr.ext.hierarchy.impl.HierarchyConfig$Permission">
                    <field name="identity">
                      <string>any</string>
                    </field>
                    <field name="read">
                      <string>true</string>
                    </field>
                    <field name="addNode">
                      <string>false</string>
                    </field>
                    <field name="setProperty">
                      <string>true</string>
                    </field>
                    <field name="remove">
                      <string>false</string>
                    </field>
                  </object>
                </value>
              </collection>
            </field>
          </object>
        </value>

      </collection>
    </field>
```

```
      </object>
    </object-param>
  </init-params>
</component-plugin>
```

Roles Configuration

Overview

The roles plug-in component defines roles in Forum of eXo Platform 3. This convenient application defines access to a set of functions within the application. Currently, it only defines the person who has the *administrator* role. Administrators can get access to administration functions. At runtime, the application gets data from the roles plug-in to decide whether the logged user has the administrative role or not.

Configuration

The plug-in is configured in the *roles-configuration.xml* file.

When the user signs in, his or her username, group and membership will be compared with the user roles defined in the *.xml* file that is provided by the roles plug-in component.

In particular, at runtime of ForumService, the roles plug-in component is called. The role plug-in is configured in the *roles-configuration.xml* file. The plug-in component named *add.role.rules.plugin* will be referred to *org.exoplatform.ks.common.conf.RoleRulesPlugin* to create users for Forum corresponding to users who exist in the organization database. In addition, the list of users who have *administration* roles are also defined.

```
<component-plugin>
  <name>add.role.rules.plugin</name>
  <set-method>addRolePlugin</set-method>
  <type>org.exoplatform.ks.common.conf.RoleRulesPlugin</type>
  <description>add role rules</description>
  <init-params>
    <value-param>
      <name>role</name>
      <description>name of the role</description>
      <value>ADMIN</value>
    </value-param>
    <values-param>
      <name>rules</name>
      <description>rules of the role</description>
      <value>root</value>
```

```
      <!--value>admin</value -->
      <!--value>member:/demo</value-->
      <!--value>/forums/admin</value-->
      <!--value>*:/forum/admin</value-->
      <!--value>/platform/administrators</value-->
      <!--value>manager:/platform/users</value-->
      <!--value>*:/somegroup/somesubgroup</value-->
      <!--value>manager:/somegroup/someothergroup</value-->
    </values-param>
  </init-params>
</component-plugin>
```

In which:

Name	Set-method	Type	Description
add.role.rules.plugin	addRolePlugin	org.exoplatform.ks.common.conf.RoleRulesPlugin	Add role rules.

- Init-params:

Name	Possible value	Default value	Description
role	`string`	`ADMIN`	The name of role.
rules	`string`	`root`	The rules of role.

- When the *role-configuration.xml* file is executed, the administration role (with `ADMIN` value) will be checked and assigned to a matrix of *users/groups/memberships* defined inside the "value" tags as below:

```
<value>...</value>
```

For example:

```
...
<value>root</value>
<value>john</value>
<value>/platform/administrators</value>
<value>member:/VIP</value>
```

```
<value>validator:/VIP</value>
...
```

In the example above, the default administrators of Forum include *root*, *john*, users in */platform/administrators* group and users who have *member/validator* memberships in the VIP group.

When being *root*, the users who belong to the */platform/administrators* group or who have *member/validator* memberships in the VIP group and sign in the Forum, they will be identified as the default administrator of Forum.

To add or remove the default administrator of the Forum, simply edit the *roles-configuration.xml* file, add or remove the relevant "value" tags.

```
...
<values-param>
  ...
  <value>...</value>
  ...
</values-param>
...
```

The default administrators of the Forum can only change their roles by editing in the *roles-configuration.xml* file.

At runtime, modifications in the *roles-configuration.xml* file will be read and database will be updated. Normal users of the Forum and default administration will be created correspondingly.

ProfileProvider Configuration

Overview

Forum and FAQ applications are to show some information about posters. The way to retrieve that information is pluggable through the *ContactProvider* component.

For public internet websites, users can provide personal information, such as personal email address and location. To enable, simply override the *ContactProvider* component in your configuration.

Configuration

Configure the *profile-configuration.xml* file as shown below:

```
```

```
<component>
  <key>org.exoplatform.ks.common.user.ContactProvider</key>
  <type>org.exoplatform.ks.common.user.DefaultContactProvider</type>
  <!--<type>org.exoplatform.ks.ext.common.SocialContactProvider</type> -->
</component>
```

When eXo Knowledge is integareted in eXo Platform and if you want to use ProfileProvider from eXo Social, you need to change "**type**" *org.exoplatform.ks.common.user.DefaultContactProvider* into *org.exoplatform.ks.ext.common.SocialContactProvider*.

Use ContactProvider

You can get the *ContactProvider* as follows:

```
public CommonContact getPersonalContact(String userId) throws Exception {
  try {
    if(userId.indexOf(Utils.DELETED) > 0) return new CommonContact();
                ContactProvider provider = (ContactProvider)
 PortalContainer.getComponent(ContactProvider.class) ;
 return provider.getCommonContact(userId);
  } catch (Exception e) {
 return new CommonContact();
  }
}
```

In eXo Knowlege, when using ContactProvider, you can use one of two following classes:

- *DefaultContactProvider*
- *SocialContactProvider*

By DefaultContactProvider

When you start the Tomcat, the *DefaultContactProvider* class will be initialized and the *OrganizationService* component is set within the *DefaultContactProvider* component.

The *DefaultContactProvider* class allows you to get user information via the *OrganizationService* component.

```
public CommonContact getCommonContact(String userId) {
```

```
 CommonContact contact = new CommonContact();
 try {
   User user = orgService.getUserHandler().findUserByName(userId);
   UserProfile profile = orgService.getUserProfileHandler().findUserProfileByName(userId);
   contact.setEmailAddress(user.getEmail());
   contact.setFirstName(user.getFirstName());
   contact.setLastName(user.getLastName());
   if(profile.getUserInfoMap() != null) {
     contact.setAvatarUrl(profile.getAttribute("user.other-info.avatar.url"));
     contact.setBirthday(profile.getAttribute("user.bdate"));
     contact.setCity(profile.getAttribute("user.home-info.postal.city"));
     contact.setCountry(profile.getAttribute("user.home-info.postal.country"));
     contact.setGender(profile.getAttribute("user.gender"));
     contact.setJob(profile.getAttribute("user.jobtitle"));
     contact.setMobile(profile.getAttribute("user.home-info.telecom.mobile.number"));
     contact.setPhone(profile.getAttribute("user.business-info.telecom.telephone.number"));
     contact.setWebSite(profile.getAttribute("user.home-info.online.uri"));
   }
 } catch (Exception e) {
   log.error("Could not retrieve forum user profile for " + userId + ": " ,e);
 }
 return contact;
}
```

- The information which is get by the user includes:

Name	Type	Description
email	`String`	Email of user.
firstName	`String`	First name of user.
lastName	`String`	Last name of user.

- The information which is get via *UserProfile* includes:

Attribute	Type	Description
user.other-info.avatar.url	`String`	The path containing the user's avatar.
user.bdate	`String`	The user's birthday.
user.home-info.postal.city	`String`	The home city of user.
user.home-info.postal.country	`String`	The home country of user.

Attribute	Type	Description
user.gender	`String`	The user's gender.
user.jobtitle	`String`	The user's job.
user.home-info.telecom.mobile.number	`String`	The home phone number of user.
user.business-info.telecom.telephone.number	`String`	The mobile number of user.
user.home-info.online.uri	`String`	The individual websites of user.

By SocialContactProvider

The *SocailContactProvider* class gets users' profiles by userId via the *IdentityManager* class.

```
public CommonContact getCommonContact(String userId) {
  CommonContact contact = new CommonContact();
  try {
                                    IdentityManager identityM = (IdentityManager)
 PortalContainer.getInstance().getComponentInstanceOfType(IdentityManager.class);
    Identity userIdentity = identityM.getIdentity(OrganizationIdentityProvider.NAME, userId, true);
    Profile profile = userIdentity.getProfile();
    if (profile.contains(Profile.EMAIL)) {
      contact.setEmailAddress(profile.getProperty(Profile.EMAIL).toString());
    }
    if (profile.contains(Profile.FIRST_NAME)) {
      contact.setFirstName(profile.getProperty(Profile.FIRST_NAME).toString());
    }
    if (profile.contains(Profile.LAST_NAME)) {
      contact.setLastName(profile.getProperty(Profile.LAST_NAME).toString());
    }
    contact.setAvatarUrl(profile.getAvatarImageSource());
    if (profile.contains(Profile.GENDER)) {
      contact.setGender(profile.getProperty(Profile.GENDER).toString());
    }

    if (profile.contains(Profile.CONTACT_PHONES)) {
      contact.setPhone(profile.getProperty(Profile.CONTACT_PHONES).toString());
    }
    if (profile.contains(Profile.URL)) {
      contact.setWebSite(profile.getProperty(Profile.URL).toString());
    }
  } catch (Exception e) {
```

```
   if (LOG.isErrorEnabled()) LOG.error(String.format("can not load contact from eXo Social Profile
 with user [%s]", userId), e);
  }
  return contact;
}
```

Forum Configuration

BBCode Configuration

Overview

The BBCode plug-in component defines default BBCode data in the *.xml* file, including BBCode tags, for example, I, B, U, SIZE, COLOR.

When the BBCode Service runs, it will get values returned from the BBCode plug-in component to initialize default BBCode data.

Configuration

Default BBCode data

The default BBCode data is configured in the *bbcodes-configuration.xml* file.

In particular, at runtime of BBCode Service, the BBCode plug-in component is called. Then, the *bbcodes-configuration.xml* file will be executed, and the component-plugin named *registerBBCodePlugin* will be referred to *org.exoplatform.ks.bbcode.spi.BBCodePlugin* to execute some objects that will generate default data.

```
<component-plugin>
  <name>forum.default.bbcodes</name>
  <set-method>registerBBCodePlugin</set-method>
  <type>org.exoplatform.ks.bbcode.spi.BBCodePlugin</type>
  <description>default supported BBCodes</description>
  <init-params>
    <object-param>
      <name>I</name>
      <description>set text in italic</description>
      <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
        <field name="tagName">
          <string>I</string>
        </field>
        <field name="replacement">
          <string>&lt;i&gt;{param}&lt;/i&gt;</string>
        </field>
```

```
      <field name="description">
        <string>Set text in italic</string>
      </field>
      <field name="example">
        <string>[I]This text is italic[/I]</string>
      </field>
      <field name="isOption">
        <string>false</string>
      </field>
      <field name="isActive">
        <string>true</string>
      </field>
    </object>
  </object-param>

  <object-param>
    <name>B</name>
    <description/>
    <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
      <field name="tagName">
        <string>B</string>
      </field>
      <field name="replacement">
        <string>&lt;b&gt;{param}&lt;/b&gt;</string>
      </field>
      <field name="description">
        <string>Set text in bold</string>
      </field>
      <field name="example">
        <string>[B]This text is bold[/B]</string>
      </field>
      <field name="isOption">
        <string>false</string>
      </field>
      <field name="isActive">
        <string>true</string>
      </field>
    </object>
  </object-param>

  <object-param>
    <name>U</name>
    <description/>
    <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
```

```
      <field name="tagName">
        <string>U</string>
      </field>
      <field name="replacement">
        <string>&lt;u&gt;{param}&lt;/u&gt;</string>
      </field>
      <field name="description">
        <string>Set text in underline</string>
      </field>
      <field name="example">
        <string>[U]This text is underline[/U]</string>
      </field>
      <field name="isOption">
        <string>false</string>
      </field>
      <field name="isActive">
        <string>true</string>
      </field>
    </object>
  </object-param>

  <object-param>
    <name>COLOR</name>
    <description/>
    <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
      <field name="tagName">
        <string>COLOR</string>
      </field>
      <field name="replacement">
        <string>&lt;font color="{option}"&gt;{param}&lt;/font&gt;</string>
      </field>
      <field name="description">
        <string>The [color=option] tag allows you to change the color of your text.</string>
      </field>
      <field name="example">
        <string>[COLOR=blue]This text is blue[/COLOR]</string>
      </field>
      <field name="isOption">
        <string>true</string>
      </field>
      <field name="isActive">
        <string>true</string>
      </field>
    </object>
```

```
  </object-param>

  <object-param>
    <name>SIZE</name>
    <description/>
    <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
      <field name="tagName">
        <string>SIZE</string>
      </field>
      <field name="replacement">
        <string>&lt;font size="{option}"&gt;{param}&lt;/font&gt;</string>
      </field>
      <field name="description">
        <string>The [size=option] tag allows you to change the size of your text.</string>
      </field>
      <field name="example">
        <string>[size=+2]this text is two sizes larger than normal[/size]</string>
      </field>
      <field name="isOption">
        <string>true</string>
      </field>
      <field name="isActive">
        <string>true</string>
      </field>
    </object>
  </object-param>

  <object-param>
    <name>FONT</name>
    <description/>
    <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
      <field name="tagName">
        <string>FONT</string>
      </field>
      <field name="replacement">
        <string>&lt;font face="{option}"&gt;{param}&lt;/font&gt;</string>
      </field>
      <field name="description">
        <string>The [font=option] tag allows you to change the font of your text.</string>
      </field>
      <field name="example">
        <string>[font=courier]this text is in the courier font[/font]</string>
      </field>
      <field name="isOption">
```

```
      <string>true</string>
    </field>
    <field name="isActive">
      <string>true</string>
    </field>
  </object>
</object-param>

<object-param>
  <name>HIGHLIGHT</name>
  <description/>
  <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
    <field name="tagName">
      <string>HIGHLIGHT</string>
    </field>
    <field name="replacement">
      <string>&lt;span style="font-weight: bold; color: blue;"&gt;{param}&lt;/span&gt;</string>
    </field>
    <field name="description">
      <string>The [highlight] tag allows you to make highlight of your text.</string>
    </field>
    <field name="example">
      <string>[highlight]this text is highlighted[/highlight]</string>
    </field>
    <field name="isOption">
      <string>false</string>
    </field>
    <field name="isActive">
      <string>true</string>
    </field>
  </object>
</object-param>

<object-param>
  <name>LEFT</name>
  <description/>
  <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
    <field name="tagName">
      <string>LEFT</string>
    </field>
    <field name="replacement">
      <string>&lt;div align="left"&gt;{param}&lt;/div&gt;</string>
    </field>
    <field name="description">
```

```
        <string>The [left] tag allows alignment text to left. </string>
      </field>
      <field name="example">
        <string>[LEFT]This text is left-aligned[/LEFT]</string>
      </field>
      <field name="isOption">
        <string>false</string>
      </field>
      <field name="isActive">
        <string>true</string>
      </field>
    </object>
  </object-param>

  <object-param>
    <name>RIGHT</name>
    <description/>
    <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
      <field name="tagName">
        <string>RIGHT</string>
      </field>
      <field name="replacement">
        <string>&lt;div align="right"&gt;{param}&lt;/div&gt;</string>
      </field>
      <field name="description">
        <string>The [right] tag allows alignment text to right.</string>
      </field>
      <field name="example">
        <string>[RIGHT]example[/RIGHT]</string>
      </field>
      <field name="isOption">
        <string>[RIGHT]this text is right-aligned[/RIGHT]</string>
      </field>
      <field name="isActive">
        <string>true</string>
      </field>
    </object>
  </object-param>

  <object-param>
    <name>CENTER</name>
    <description/>
    <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
      <field name="tagName">
```

```
      <string>CENTER</string>
    </field>
    <field name="replacement">
      <string>&lt;div align="center"&gt;{param}&lt;/div&gt;</string>
    </field>
    <field name="description">
      <string>The [center] allows aligntment text to center.</string>
    </field>
    <field name="example">
      <string>[CENTER]this text is center-aligned[/CENTER]</string>
    </field>
    <field name="isOption">
      <string>false</string>
    </field>
    <field name="isActive">
      <string>true</string>
    </field>
  </object>
</object-param>

<object-param>
  <name>JUSTIFY</name>
  <description/>
  <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
    <field name="tagName">
      <string>JUSTIFY</string>
    </field>
    <field name="replacement">
      <string>&lt;div align="justify"&gt;{param}&lt;/div&gt;</string>
    </field>
    <field name="description">
      <string>The [justify] tag allows aligntment text to justify.</string>
    </field>
    <field name="example">
      <string>[JUSTIFY]this text is justify-aligned[/JUSTIFY]</string>
    </field>
    <field name="isOption">
      <string>false</string>
    </field>
    <field name="isActive">
      <string>true</string>
    </field>
  </object>
</object-param>
```

```
  <object-param>
    <name>EMAIL</name>
    <description/>
    <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
      <field name="tagName">
        <string>EMAIL</string>
      </field>
      <field name="replacement">
        <string>&lt;a href="mailto:{param}"&gt;{param}&lt;/a&gt;</string>
      </field>
      <field name="description">
        <string>The [email] tag allows you to link to an email address.</string>
      </field>
      <field name="example">
        <string>[email]demo@example.com[/email]</string>
      </field>
      <field name="isOption">
        <string>false</string>
      </field>
      <field name="isActive">
        <string>true</string>
      </field>
    </object>
  </object-param>

  <object-param>
    <name>EMAIL-OPT</name>
    <description/>
    <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
      <field name="tagName">
        <string>EMAIL</string>
      </field>
      <field name="replacement">
        <string>&lt;a href="mailto:{option}"&gt;{param}&lt;/a&gt;</string>
      </field>
      <field name="description">
      <string>The [email=option] tag allows link to an email address and use an optional parameter
to 'name' of this link.</string>
      </field>
      <field name="example">
        <string>[email=demo@example.com]Click Here to Email me[/email] </string>
      </field>
      <field name="isOption">
```

```
      <string>true</string>
    </field>
    <field name="isActive">
      <string>true</string>
    </field>
  </object>
</object-param>

<object-param>
  <name>URL</name>
  <description/>
  <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
    <field name="tagName">
      <string>URL</string>
    </field>
    <field name="replacement">
      <string>&lt;a target='_blank' href="{param}"&gt;{param}&lt;/a&gt;</string>
    </field>
    <field name="description">
      <string>The [url] tag allows link to other websites and files.</string>
    </field>
    <field name="example">
      <string>[URL]http://www.exoplatform.com[/URL]</string>
    </field>
    <field name="isOption">
      <string>false</string>
    </field>
    <field name="isActive">
      <string>true</string>
    </field>
  </object>
</object-param>

<object-param>
  <name>URL-OPT</name>
  <description/>
  <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
    <field name="tagName">
      <string>URL</string>
    </field>
    <field name="replacement">
      <string>&lt;a target='_blank' href="{option}"&gt;{param}&lt;/a&gt;</string>
    </field>
    <field name="description">
```

```
          <string>The [url=option] tag allows link to other websites and files and usean optional parameter to 'name' of thislink.</string>
        </field>
        <field name="example">
          <string>[URL=http://www.exoplatform.com]Click goto exoplatform website.[/URL]</string>
        </field>
        <field name="isOption">
          <string>true</string>
        </field>
        <field name="isActive">
          <string>true</string>
        </field>
      </object>
    </object-param>

    <object-param>
      <name>GOTO</name>
      <description/>
      <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
        <field name="tagName">
          <string>GOTO</string>
        </field>
        <field name="replacement">
          <string>&lt;a href="{option}"&gt;{param}&lt;/a&gt;</string>
        </field>
        <field name="description">
          <string>Allows goto directly to link instead of open a new window or a new tab. </string>
        </field>
        <field name="example">
          <string>[goto=http://www.exoplatform.com]Goto this link.[/goto]&gt;</string>
        </field>
        <field name="isOption">
          <string>true</string>
        </field>
        <field name="isActive">
          <string>true</string>
        </field>
      </object>
    </object-param>

    <object-param>
      <name>LIST</name>
      <description/>
      <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
```

```
      <field name="tagName">
        <string>LIST</string>
      </field>
      <field name="replacement">
        <string>You can not define this bbcode tag. It is defined by the developer.</string>
      </field>
      <field name="description">
        <string>The [list] tag allows create simple, each bullet is denoted by the [*] tag.</string>
      </field>
      <field name="example">
        <string>[list][*]list item 1[*]list item 2[/list]</string>
      </field>
      <field name="isOption">
        <string>false</string>
      </field>
      <field name="isActive">
        <string>true</string>
      </field>
    </object>
  </object-param>

  <object-param>
    <name>LIST-OPT</name>
    <description/>
    <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
      <field name="tagName">
        <string>LIST</string>
      </field>
      <field name="replacement">
        <string>You can not define this bbcode tag. It is defined by the developer.</string>
      </field>
      <field name="description">
          <string>The [list=option] tag allows create bulleted lists specifying an option. Within the
value portion, each bullet is denoted by the [*] tag.</string>
      </field>
      <field name="example">
        <string>[list=1][*]list item 1[*]list item 2[/list]</string>
      </field>
      <field name="isOption">
        <string>true</string>
      </field>
      <field name="isActive">
        <string>true</string>
      </field>
```

```
      </object>
    </object-param>

    <object-param>
      <name>IMG</name>
      <description/>
      <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
        <field name="tagName">
          <string>IMG</string>
        </field>
        <field name="replacement">
          <string>&lt;img border="0" alt="" src="{param}" class="inlineimg"/&gt;</string>
        </field>
        <field name="description">
          <string>The [img] tag allows you to shows the image indicated by {url}</string>
        </field>
        <field name="example">
          <string>[url=http://www.google.com.vn] [img]http://groups.google.com.vn/groups/img/3nb/
groups_medium_vi.gif[/img] [/url]</string>
        </field>
        <field name="isOption">
          <string>false</string>
        </field>
        <field name="isActive">
          <string>true</string>
        </field>
      </object>
    </object-param>

    <object-param>
      <name>QUOTE</name>
      <description/>
      <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
        <field name="tagName">
          <string>QUOTE</string>
        </field>
        <field name="replacement">
         <string>&lt;div style="background:#ededf7; border:1px solid #d8d8d8; padding:6px 6px 6px
 15px; margin:2px 0px;"&gt;{param}&lt;/div&gt;</string>
        </field>
        <field name="description">
          <string>The [quote] tag allows attribute content of post.</string>
        </field>
        <field name="example">
```

```
        <string>[quote]Lorem ipsum dolor sit amet[/quote]</string>
      </field>
      <field name="isOption">
        <string>false</string>
      </field>
      <field name="isActive">
        <string>true</string>
      </field>
    </object>
  </object-param>

  <object-param>
    <name>QUOTE-OPT</name>
    <description/>
    <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
      <field name="tagName">
        <string>QUOTE</string>
      </field>
      <field name="replacement">
            <string>&lt;div style="background:#ededf7; border:1px solid #d8d8d8; padding:6px
 6px 6px 15px; margin:2px 0px;"&gt;&lt;div&gt;Originally Posted by &lt;strong&gt;{option}&lt;/
strong&gt;&lt;/div&gt;&lt;div&gt;{param}&lt;/div&gt;&lt;/div&gt;</string>
      </field>
      <field name="description">
        <string>The [quote=option] tag allows attibute content and user name of poster.</string>
      </field>
      <field name="example">
        <string>[quote=John Doe]Lorem ipsum dolor sit amet[/quote]</string>
      </field>
      <field name="isOption">
        <string>true</string>
      </field>
      <field name="isActive">
        <string>true</string>
      </field>
    </object>
  </object-param>

  <object-param>
    <name>CODE</name>
    <description/>
    <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
      <field name="tagName">
        <string>CODE</string>
```

```
      </field>
      <field name="replacement">
            <string>&lt;div style="background:#ededed; border:1px inset #7b7b7b; margin:5px;
 overflow:auto;"&gt;&lt;pre style="margin: 0px; padding: 0px; overflow: auto; text-align: left;"
dir="ltr"&gt;&lt;div&gt;{param}&lt;/div&gt;&lt;/pre&gt;&lt;/div&gt;</string>
      </field>
      <field name="description">
        <string>The [code] tag allows you to view source code html.</string>
      </field>
      <field name="example">
        <string>[code]&lt;div&gt;some text or code html&lt;/div&gt;[/code]</string>
      </field>
      <field name="isOption">
        <string>false</string>
      </field>
      <field name="isActive">
        <string>true</string>
      </field>
    </object>
  </object-param>

  <object-param>
    <name>CSS</name>
    <description/>
    <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
      <field name="tagName">
        <string>CSS</string>
      </field>
      <field name="replacement">
        <string>&lt;span class="{option}"&gt;{param}&lt;/span&gt;</string>
      </field>
      <field name="description">
      <string>The [css=option] tag allows you to add div tag and set class Name for this it.</string>
      </field>
      <field name="example">
        <string>[css=highlight]Text is highlight[/css]</string>
      </field>
      <field name="isOption">
        <string>true</string>
      </field>
      <field name="isActive">
        <string>true</string>
      </field>
    </object>
```

```
    </object-param>

    <object-param>
      <name>SLIDESHARE</name>
      <description/>
      <object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
        <field name="tagName">
          <string>SLIDESHARE</string>
        </field>
        <field name="replacement">
          <string>
          &lt;div style="width:425px; height:355px;" align="center"&gt;
          &lt;object style="margin:0px" width="425" height="355"&gt;
                                &lt;param name="movie" value="http://static.slidesharecdn.com/swf/
ssplayer2.swf?doc={option}&amp;rel=0"/&gt;
            &lt;param name="allowFullScreen" value="true"/&gt;
            &lt;param name="allowScriptAccess" value="always"/&gt;
         &lt;embed src="http://static.slidesharecdn.com/swf/ssplayer2.swf?doc={option}&amp;rel=0"
        type="application/x-shockwave-flash" allowscriptaccess="always" allowfullscreen="true"
 width="425" height="355"&gt;
            &lt;/embed&gt;
          &lt;/object&gt;
          &lt;b&gt;{param}&lt;/b&gt;&lt;/div&gt;
          </string>
        </field>
        <field name="description">
              <string>The [SLIDESHARE=option] tag allows you to run slide in slidesharecdn.com
 site.</string>
        </field>
        <field name="example">
          <string>[SLIDESHARE=slideId]My slide[/SLIDESHARE]</string>
        </field>
        <field name="isOption">
          <string>true</string>
        </field>
        <field name="isActive">
          <string>true</string>
        </field>
      </object>
    </object-param>
  </init-params>
</component-plugin>
```

- In which,

Name	Set method	Type	Description
forum.default.bbcodes	registerBBCodePlugin	org.exoplatform.ks.bbcode.spi.BBCodePlugin	Define formats for data displayed on UI.

- The BBCode array is defined by the *org.exoplatform.ks.bbcode.spi.BBCodeData* object as below:

```
<object type="org.exoplatform.ks.bbcode.spi.BBCodeData">
  <field name="tagName">
    <string>I</string>
  </field>
  <field name="replacement">
    <string>&lt;i&gt;{param}&lt;/i&gt;</string>
  </field>
  <field name="description">
    <string>Set text in italic</string>
  </field>
  <field name="example">
    <string>[I]This text is italic[/I]</string>
  </field>
  <field name="isOption">
    <string>false</string>
  </field>
  <field name="isActive">
    <string>true</string>
  </field>
</object>
```

- The BBCode includes basic data which are defined in the field tag with a specific name as below:

```
<field name="tagName">
  <string>I</string>
</field>
<field name="replacement">
  <string>&lt;i&gt;{param}&lt;/i&gt;</string>
</field>
<field name="description">
  <string>Set text in italic</string>
</field>
```

```
<field name="example">
  <string>[I]This text is italic[/I]</string>
</field>
<field name="isOption">
  <string>false</string>
</field>
<field name="isActive">
  <string>true</string>
</field>
```

In which:

Field name	Value	Description
tagName	`string`	The text for the BBCode, which is put between two square brackets ([ ]). For example, for the bold tag, if you type [b], the BBCode tag will be **b** without any square brackets ([ ]).
replacement	`string`	The HTML code that replaces the BBCode entered by the user. Make sure that you include '{param}' (without quotes) to insert the text between opening and closing BBCode tags, and '{option}' for the parameter within the BBCode tag. You can only use *option* if 'Use Option' is selected.
description	`string`	The piece of text to describe the BBCode tag, including HTML tags if you want.
example	`string`	The sample piece of BBCode to use as an example for the particular BBCode. For example, to demonstrate the usage of [b] tag, enter [b]text[/b].
isOption	`true`, `false`	Select the [tag=option] [/tag] style tag, rather than just a [tag][/tag]style tag. This

Field name	Value	Description
		function will be created if you select this option.
isActive	`true`, `false`	Activate the BBCode tag.

Forums Initializer

Overview

The Initialization plug-in component defines the default Forum data in the **.xml** file, including categories, forums, topics and posts.

When the Forum Service runs, it will get values which are returned from the Initialization plug-in component to initialize default data of the Forum.

Configuration

Default forum data

The default forum data is configured in the *war:webapp/WEB-INF/conf/ksdemo/ks/services-configuration.xml* file.

In particular, when the ForumService starts, the Initialization plug-in component is called. Next, the *services-configuration.xml* file is executed. The component-plugin named *addInitialDefaultDataPlugin* will refer to *org.exoplatform.forum.service.conf.InitializeForumPlugin* to execute some objects to create default data for the Forum application.

```
<component-plugin>
  <name>default.data</name>
  <set-method>addInitialDefaultDataPlugin</set-method>
  <type>org.exoplatform.forum.service.conf.InitializeForumPlugin</type>
  <description>description</description>
  <init-params>
    <object-param>
      <name>livedemo.default.configuration</name>
      <description>initial data for live demo</description>
      <object type="org.exoplatform.forum.service.conf.ForumInitialData">
        <field name="categories">
          <collection type="java.util.ArrayList">
            <value>
              <object type="org.exoplatform.forum.service.conf.CategoryData">
                <field name="owner">
                  <string>root</string>
                </field>
                <field name="name">
```

```
            <string>Knowledge Suite</string>
          </field>
          <field name="description">
            <string>All about eXo KS</string>
          </field>
          <field name="forums">
            <collection type="java.util.ArrayList">
              <value>
                <object type="org.exoplatform.forum.service.conf.ForumData">
                  <field name="owner"><string>root</string></field>
                  <field name="name"><string>Live demo</string></field>
                  <field name="description"><string>Questions about this demo</string></field>
                  <field name="topics">
                    <collection type="java.util.ArrayList">
                      <value>
                        <object type="org.exoplatform.forum.service.conf.TopicData">
                          <field name="name"><string>Demo data policy</string></field>
                          <field name="icon"><string>Shield</string></field>
                          <field name="owner"><string>root</string></field>
                          <field name="content"><string>
                          Welcome to eXo Knowledge Suite live demo!
                          We hope you enjoy discovering eXo Forum and FAQ applications features.
                          You don't need to be logged in to see the applications in action.
                          But the power of KS lies in the rich set of admin/moderation features.
                    We didn't want you to miss them so, when you [b][url="/portal/public/classic/register"]
                          create a demo account[/url][/b], you will be granted full permissions.
                        Anybody can become an administrator or a moderator and play in the sandbox!
                                As a consequence, the data for this forum (including the accounts) is
 not meant to stay.
                          [b][center]WE MAY RESET FORUMS AND FAQS ANYTIME[/center][/b]
                              Enjoy and don't forget to send feedback at [email]ks@exoplatform.com[/
email]</string></field>
                        </object>
                      </value>
                    </collection>
                  </field>
                </object>
              </value>
            </collection>
          </field>
        </object>
      </value>
    </collection>
  </field>
```

```
    </object>
   </object-param>
  </init-params>
</component-plugin>
```

- In which,

Name	Set-method	Type	Description
default.data	`addInitialDataPlugin`	`org.exoplatform.forum.service.conf.InitializeForumPlugin`	The initial default data of Forum.

- Init-param

Name	Possible value	Default value	Description
livedemo.default.configuration	`object`	`org.exoplatform.forum.service.conf.ForumInitialData`	The initial data for live demo.

- Category array

After the *org.exoplatform.forum.service.conf.InitializeForumPlugin* object has been executed, the *org.exoplatform.forum.service.conf.ForumInitialData* object will be called. It returns a category array. The value of category array is defined by the *org.exoplatform.forum.service.conf.CategoryData* object as below:

```
<object-param>
  <name>livedemo.default.configuration</name>
  <description>initial data for live demo</description>
  <object type="org.exoplatform.forum.service.conf.ForumInitialData">
    <field name="categories">
      <collection type="java.util.ArrayList">
        <value>
          <object type="org.exoplatform.forum.service.conf.CategoryData">
            <field name="owner">
              <string>root</string>
            </field>
            <field name="name">
```

```
            <string>Knowledge Suite</string>
          </field>
          <field name="description">
            <string>All about eXo KS</string>
          </field>
          ...
        </object>
      </value>
    </collection>
  </field>
 </object>
</object-param>
```

1. Category includes some basic data which are defined in the **field** tag with a specific name as below:

```
<field name="owner">
 <string>root</string>
</field>
<field name="name">
 <string>Knowledge Suite</string>
</field>
<field name="description">
 <string>All about eXo KS</string>
</field>
```

In which:

Field	Possible value	Default value	Description
owner	`user id`	`root`	The creator of Category.
name	`string`	`Knowledge Suite`	The title of Category.
description	`string`	`All about eXo KS`	The brief description of Category.

- Modify values of Category

Values of the default Category can be changed by editing text values in the *string* tag of each *field* by the other one. In the sample code above, the *org.exoplatform.forum.service.conf.CategoryData* object is called. It means that only one default Category is defined. If you want to define more

default Categories, repeat calling the *org.exoplatform.forum.service.conf.CategoryData* object and define values for the new Category with the sample code as below:

```
<value>
  <object type="org.exoplatform.forum.service.conf.CategoryData">
    <field name=" ">
      ...
    </field>
  </object>
</value>
```

Forum array

Category may contain one or more Forums. The value of the Forum is defined in the **forums** field. It returns a forum array. The value of forum array is defined by the **org.exoplatform.forum.service.conf.ForumData** object as below:

```
<field name="forums">
  <collection type="java.util.ArrayList">
    <value>
      <object type="org.exoplatform.forum.service.conf.ForumData">
        <field name="owner"><string>root</string></field>
        <field name="name"><string>Live demo</string></field>
        <field name="description"><string>Questions about this demo</string></field>
        ...
      </object>
    </value>
  </collection>
</field>
```

- Basic Forum data

Forum includes some basic data which are defined in the **field** tag with the specific name as above.

In which:

Field	Possible value	Default value	Description
owner	`user id`	`root`	The creator of default Forum.
name	`string`	`Live demo`	The name or title of default Forum.

Field	Possible value	Default value	Description
description	`string`	`Questions about this demo`	The brief description of default Forum.

The default Forum values can be changed by editing text values in the *string* tag of each *field* by the other one.

In the sample code above, the *org.exoplatform.forum.service.conf.ForumData* object is called only one time. It means that only one default Forum is defined inside the default Category named *Knowledge Suite*. If you want to define more default Forums, repeat calling the *org.exoplatform.forum.service.conf.ForumData* object and define values for the new Forum with the sample code as below:

```
<value>
  <object type="org.exoplatform.forum.service.conf.ForumData">
    <field name=" ">
      ...
    </field>
  </object>
</value>
```

Forum topics

1. Forum may contain one or more topics. The value of topic is defined in the **topics** field. It returns a topic array. The value of topic array is defined by the *org.exoplatform.forum.service.conf.TopicData* object as below:

```
<field name="topics">
  <collection type="java.util.ArrayList">
    <value>
      <object type="org.exoplatform.forum.service.conf.TopicData">
        <field name="name"><string>Demo data policy</string></field>
        <field name="icon"><string>Tux</string></field>
        <field name="owner"><string>root</string></field>
        <field name="content"><string>Welcome to eXo Forum live demo. ... at anytime.</string></
field>
        ...
      </object>
    </value>
  </collection>
```

```
</field>
```

A topic includes some basic data which are defined in the *field* tag with a specific name as above.

In which:

Field	Possible value	Default value	Description
name	`string`	`Demo data policy`	The name or title of default topic.
icon	`string`	`Tux`	The default icon of default topic.
owner	`user id`	`root`	The creator of default topic.
content	`string`	`Welcome to eXo Forum live demo...at anytime.`	The main content of default topic.

The values of default topic can be changed by editing text values in the *string* tag of each *field*.

With the sample code above, the *org.exoplatform.forum.service.conf.TopicData* object is called only one time. It means that only one default topic is defined inside the default Forum named *Live demo*. If you want to define more default topics, repeat calling the *org.exoplatform.forum.service.conf.TopicData* object and define values for the new topic with the sample code as below:

```
<value>
  <object type="org.exoplatform.forum.service.conf.TopicData">
    <field name=" ">
      ...
    </field>
  </object>
</value>
```

1. Topic may contain one or more posts. The value of the post is defined in the *posts* field. It returns a post array. The value of the post array is defined by the *org.exoplatform.forum.service.conf.PostData* object as below:

```
<field name="posts">
  <collection type="java.util.ArrayList">
    <value>
```

```
    <object type="org.exoplatform.forum.service.conf.PostData">
      <field name="name"><string>Reply: Demo data policy</string></field>
      <field name="icon"><string>IconsView</string></field>
      <field name="owner"><string>root</string></field>
            <field name="content"><string>Enjoy and don't forget to send feedback at
ks@exoplatform.com</string></field>
      ...
    </object>
  </value>
 </collection>
</field>
```

A post includes some basic data which are defined in the *field* tag with a specific name as above.

In which:

Field	Possible value	Default value	Description
name	`string`	`Reply: Demo data policy`	The name or title of default post.
icon	`string`	`IconsView`	The default icon of default post.
ownwer	`user id`	`root`	The creator of default post.
content	`string`	`Enjoy and don't forget to send feedback at ks@exoplatform.com`	The main content of default post.

The default post values can be changed by editing text values in the *string* tag of each *field*.

With the sample code above, the **org.exoplatform.forum.service.conf.PostData** object is called only one time. It means that only one default post is defined inside the default topic named **Demo data policy**. If you want to define more default posts, repeat calling the **org.exoplatform.forum.service.conf.PostData** object and define values for the new post with the sample code as below:

```
<value>
 <object type="org.exoplatform.forum.service.conf.PostData">
  <field name=" ">
   ...
  </field>
 </object>
```

```
</value>
```

By default, the default Forum data can only be changed by modifying the *services-configuration.xml* file.

At runtime, the new changes in the *services-configuration.xml* file will be executed and updated. The default Forum data will be created correspondingly.

Initial Data Plugin

The Initial Data plugin is configured in the *services-configuration.xml* file. In details, at runtime of Forum Service, the Initialization plugin component is called, the *services-configuration.xml* file will be executed. The component-plugin named *addInitialDataPlugin* will refer to *org.exoplatform.forum.service.conf.ForumInitialDataPlugin* to import some objects to create data for the Forum service. The default data in the .zip or .xml file is initialized as follows:

```
<component-plugin>
  <name>technical.forum</name>
  <set-method>addInitialDataPlugin</set-method>
  <type>org.exoplatform.forum.service.conf.ForumInitialDataPlugin</type>
  <description>Initialize</description>
  <init-params>
    <values-param>
      <name>locations</name>
      <description>location where Forum export format file is stored</description>
      <value>war:/data/forum/data-full-forum.zip</value>
      <!-- value>war:/data/forum/forumCategory.xml</value -->
    </values-param>
  </init-params>
</component-plugin>
```

- In which:

Name	Set-Method	Type	Description
technical.forum	`addInitialDataPlugin`	`org.exoplatform.forum.service.conf.ForumInitialDataPlug`	Initialize the Data plugin.

- Init-params

Name	Possible values	Default value	Description
locations	`String`	`war:/data/forum/data-full-forum.zip`	The location where the Forum export format file is stored.

Auto-prune

Overview

The Auto-prune component is to prune inactive topics which have not been viewed, edited or received for a given period. The "prune" operation does not denote to the physical removal of topics, but sets them to invisible. The function helps you not clutter busy forums from outdated information.

When the Job Scheduler runs, it will get values returned from the Auto-prune plug-in component to identify topics which have to be inactivated in the Forum application. These topics will be invisible to users.

Configuration

The properties of Auto-prune plug-in are configured in the *war:webapp/WEB-INF/ks-extension/ks/forum/prune-configuration.xml* file.

In particular, at runtime of Job Scheduler, the Auto-prune plugin component is called. Then, the *prune-configuration.xml* file will be executed. The component-plugin named *ForumDeactiveJob* will refer to *org.exoplatform.forum.service.conf.DeactivePeriodJob* to inactivate topics in Forum which meets predefined inactivation properties.

```
<component-plugin>
  <name>ForumDeactiveJob</name>
  <set-method>addPeriodJob</set-method>
  <type>org.exoplatform.forum.service.conf.DeactivePeriodJob</type>
  <description>add a Deactive job to the JobSchedulerService</description>
  <init-params>
    <properties-param>
      <name>job.info</name>
      <description>save the monitor data  periodically</description>
      <property name="jobName" value="DeactiveJob"/>
      <property name="groupName" value="KnowlegedSuite"/>
      <property name="job" value="org.exoplatform.forum.service.conf.DeactiveJob"/>
      <property name="repeatCount" value="0"/>
      <property name="period" value="7200000"/> <!-- 2 hours-->
      <property name="startTime" value="+0"/>
      <property name="endTime" value=""/>
    </properties-param>
    <properties-param>
      <name>deactive.info</name>
      <description/>
      <property name="inactiveDays" value="15"/>
      <property name="forumName" value="Live demo"/>
```

```
    </properties-param>
  </init-params>
</component-plugin>
```

- In which,

Name	Set-method	Type	Description
ForumDeactiveJob	addPeriodJob	org.exoplatform.forum.service.conf.DeactivePeriodJob	Add a DeactiveJob to the JobSchedulerService.

- The properties for the Auto-prune plug-in are defined in the *property* tag with the format as below:

```
...
<property name="jobName" value="DeactiveJob"/>
<property name="groupName" value="KnowlegedSuite"/>
<property name="job" value="org.exoplatform.forum.service.conf.DeactiveJob"/>
<property name="repeatCount" value="0"/>
<property name="period" value="7200000"/> <!-- 2 hours-->
<property name="startTime" value="+0"/>
<property name="endTime" value=""/>
...
<property name="inactiveDays" value="15"/>
<property name="forumName" value="Live demo"/>
...
```

In details:

Property name	Possible value	Default value	Description
jobname	`String`	`DeactiveJob`	The name of job which will be executed.
groupname	`String`	`KnowlegedSuite`	The name of application which will be executed.
job	`Class path`	org.exoplatform.forum.service.conf.DeactiveJob	The reference function of the job which will be executed.
repeatCount	`Long`	`0`	The repeating time for the job, meaning that

Property name	Possible value	Default value	Description
			how many times the job will be executed. The `0` value means that *DecactiveJob* is called at runtime only without repeating. If the value is set to **2** or **3**, *DecactiveJob* will be called two or three times correspondingly.
period	`Long`	`72000000`	The interval between job executions.
starttime	`Integer`	`+0`	The start time when the function executes. The *starttime* is 0, meaning that the time to start executing DecactiveJob is the runtime.
endtime	`Integer`	`null`	The end time when the function stops executing. The *endtime* is blank, meaning that there is no limitation for the end time for *DecactiveJob*.

With start and end time, you can give a specific date in the format: yyyy-mm-dd HH:mm:ss.sss to define the start and end time for *DecactiveJob*. Besides, inactive information is also defined:

Property name	Possible value	Default value	Description
inactiveDays	`Integer`	`15`	The number of days the topic has not been activated. The *inactivateDays* is set to **1**, meaning that all the topics, which have one inactivated day, will be set as inactivated status. They will be invisible.

Property name	Possible value	Default value	Description
forumname	`String`	`Live Demo`	The name of Forum which will be checked for Auto-prune. In case the value of *forumname* is blank, all forums will be checked for the Auto-prune. If the *forumname* is Live demo, only the Forum named 'Live demo' is checked for the Auto-prune.

By default, the default properties can only be changed by editing its value in the *prune-configuration.xml* file.

At runtime, the new changes in the *prune-configuration.xml* file are executed and updated. After that, the Auto-prune plug-in will be executed, depending on its properties.

User Statistics

Overview

The Auto-count Active Users component is to calculate the number of active users automatically. A user is considered as the active user only when he/she adds a topic/post in the Forum and his/her last post date matches the predefined interval time.

For example, if one user does not have any new posts after 15 days, he/she is not considered as an active user.

When the Job Scheduler runs, it will get values returned from the Auto-count Active Users plug-in component to identify the number of active users. This value is updated to Active Members information when the user views Forum statistics.

Configuration

The properties of Auto-count Active Users plug-in is configured in the *war:webapp/WEB-INF/ks-extension/ks/forum/statistics-configuration.xml* file.

In details, at runtime of Job Scheduler, the Auto-count Active Users plug-in component is called. Then, the *statistics-configuration.xml* file is executed. The component-plugin named *RecountActiveUserJob* will refer to *org.exoplatform.forum.service.conf.RecountActiveUserPeriodJob* to calculate the number of active users.

```
<component-plugin>
  <name>RecountActiveUserJob</name>
  <set-method>addPeriodJob</set-method>
  <type>org.exoplatform.forum.service.conf.RecountActiveUserPeriodJob</type>
  <description>add a RecountActiveUser job to the JobSchedulerService</description>
  <init-params>
    <properties-param>
      <name>job.info</name>
      <description>save the monitor data  periodically</description>
      <property name="jobName" value="RecountActiveUserJob"/>
      <property name="groupName" value="KnowlegedSuite"/>
      <property name="job" value="org.exoplatform.forum.service.conf.RecountActiveUserJob"/>
      <property name="repeatCount" value="0"/>
      <property name="period" value="7200000"/> <!-- 2 hours-->
      <property name="startTime" value="+0"/>
      <property name="endTime" value=""/>
    </properties-param>
    <properties-param>
      <name>RecountActiveUser.info</name>
      <description/>
      <property name="lastPost" value="15"/> <!-- users are active if have last posts in 15 day -->
    </properties-param>
  </init-params>
</component-plugin>
```

- In which,

Name	Method	Type	Description
RecountActiveUserJob	addPeriodJob	org.exoplatform.forum.service.conf.RecountActiveUserPeriodJob	Add a RecountActiveUser job to the JobSchedulerService.

- The properties for Auto-count Active Members plug-in are defined in the property tag as below:

```
...
<property name="jobName" value="RecountActiveUserJob"/>
<property name="groupName" value="KnowlegedSuite"/>
<property name="job" value="org.exoplatform.forum.service.conf.RecountActiveUserJob"/>
```

```
<property name="repeatCount" value="0"/>
<property name="period" value="7200000"/>
<property name="startTime" value="+0"/>
<property name="endTime" value=""/>
...
<property name="lastPost" value="15"/>
...
```

In which:

Property name	Possible value	Default value	Description
jobname	`String`	`RecountActiveUserJob`	The name of job which will be executed.
groupname	`String`	`KnowlegedSuite`	The name of application which will be executed.
job	`Class path`	org.exoplatform.forum.service.conf.RecountActiveUserJob	The reference function of job which will be executed.
repeatCount	`Long`	0	The number of times the job is repeated. If **repeatCount** is set to **0**, RecountActiveUserJob is called at runtime only without repeating. If the number is set to **2** or **3**, RecountActiveUserJob will be called two or three times.
period	`Long`	7200000 (millisecond) (equal to two hours)	The interval time to execute the job.
starttime	`Integer`	0	The start time when the function executes. The *starttime* is 0, meaning that the time to start executing RecountActiveUserJob is the runtime.

Property name	Possible value	Default value	Description
endtime	`Integer`	null	The end time when the function stops executing. The *endtime* is blank, meaning that there is no limitation for the end time for *RecountActiveUserJob*.

With start and end time, you can give a specific date in the format: yyyy-mm-dd HH:mm:ss.sss to define the start and end time for *RecountActiveUserJob*. The information of active time is also defined:

Property name	Possible value	Default value	Description
lastPost	`Integer`	15	The number of days that the user has added the last post. *lastPost* is 15, meaning that all users, who have any new posts within 15 days as from their last post date, are active members.

By default, the default properties can only be changed by editing its values in the *statistics-configuration.xml* file.

At runtime, the new changes in the *statistics-configuration.xml* file will be executed and updated. The Auto-count Active Users plug-in will be executed, depending on its properties.

Update Statistic Data

Overview

UpdateDataJob is used when there are abnormal changes in Forum data (such as migration). By default, UpdateDataJob is disabled at the server start up. When UpdateDataJob is running, it will calculate the statistic data in Forum to make sure that the statistic data are correct.

Configuration

The properties of Forum's UpdateDataJob is configured in */WEB-INF/ks-extension/ks/forum/statistics-configuration.xml* which is located in ks-extension webapp.

```
<component-plugin>
  <name>UpdateDataJob</name>
  <set-method>addPeriodJob</set-method>
  <type>org.exoplatform.services.scheduler.PeriodJob</type>
  <description>update topic count and post count to forum service</description>
  <init-params>
    <properties-param>
      <name>job.info</name>
      <description>save the monitor data  periodically</description>
      <property name="jobName" value="UpdateDataJob"/>
      <property name="groupName" value="KnowledgeSuite-forum"/>
      <property name="job" value="org.exoplatform.forum.service.conf.UpdateDataJob"/>
      <property name="repeatCount" value="1"/>
      <property name="period" value="30000"/>
      <property name="startTime" value="+0"/>
      <property name="endTime" value=""/>
    </properties-param>
  </init-params>
</component-plugin>
```

In which:

Name	Method	Type	Description
UpdateDataJob	addPeriodJob	org.exoplatform.services.scheduler.PeriodJob	Add UpdateDataJob to the JobSchedulerService.

- The properties for Auto-count Active Members plug-in are defined in the property tag as below:

```
<property name="jobName" value="UpdateDataJob"/>
<property name="groupName" value="KnowledgeSuite-forum"/>
<property name="job" value="org.exoplatform.forum.service.conf.UpdateDataJob"/>
<property name="repeatCount" value="1"/>
<property name="period" value="30000"/>
<property name="startTime" value="+0"/>
<property name="endTime" value=""/>
```

Property name	Possible value	Default value	Description
jobname	`String`	`UpdateDataJob`	

Property name	Possible value	Default value	Description
			The name of job which will be executed.
groupname	`String`	`KnowledgeSuite-forum`	The name of application which will be executed.
job	`Class path`	org.exoplatform.forum.service.conf.UpdateDataJob	The class of job which will be executed.
repeatCount	`Long`	1	The number of times the job is repeated. If **repeatCount** is set to **1**, RecountActiveUserJob is called at runtime only without repeating. If the number is set to **2** or **3**, RecountActiveUserJob will be called two or three times.
period	`Long`	30000 (millisecond) (equal to two hours)	The interval time to execute the job.
starttime	`Integer`	0	The start time when the function executes. The *starttime* is 0, meaning that the time to start executing RecountActiveUserJob is the runtime.
endtime	`Integer`	null	The end time when the function stops executing. The *endtime* is blank, meaning that there is no limitation for the end time for *UpdateDataJob*.

With start and end time, you can give a specific date in the format: yyyy-mm-dd HH:mm:ss.sss to define the start and end time for UpdateDataJob.

Default User Profile

Overview

The default Forum settings are a set of settings for a new account. It contains declarations of time zone, short date format, long date format, time format, maximum topics per page, maximum posts per page and flag for showing forum jump or not. The settings are simple, and users can change such settings to UI-based functions later.

Configuration

This configuration is declared in the file named *ks-configuration.xml*. Its path is *"[tomcat source]/webapps/ks-extension/WEB-INF/ks-extension/ks/ks-configuration.xml"* if you are running the tomcat and *"[project source]/extension/webapp/src/main/webapp/WEB-INF/ks-extension/ks/ks-configuration.xml"* if you are in the development phrase.

```
...
<external-component-plugins>
            <target-component>org.exoplatform.services.organization.OrganizationService</target-component>
  <component-plugin>
    ...
    <init-params>
      <properties-param>
        <name>user.profile.setting</name>
        <description>set default user profile</description>
        <property name="timeZone" value="GMT"/>
        <property name="shortDateFormat" value="MM/dd/yyyy"/>
        <property name="longDateFormat" value="DDD,MMM dd,yyyy"/>
        <property name="timeFormat" value="hh:mm a"/>
        <property name="maxTopic" value="10"/>
        <property name="maxPost" value="10"/>
        <property name="isShowForumJump" value="true"/>
      </properties-param>
    </init-params>

  </component-plugin>
</external-component-plugins>
...
```

In which:

Parameter	Possible value	Default value	Description
timeZone	`Time zone id`	`GMT`	The time zone set by user. For example: GMT, GMT-05:00, GMT+07:00, GMT+08:30 ... Visit the website: *http://java.sun.com/j2se/1.4.2/docs/api/java/util/TimeZone.html* for more details.
shortDateFormat	`Valid Java Date format`	`MM/dd/yyyy`	The format to display short information of date. Visit the website: *http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html* to ensure the exact format.)
longDateFormat	`Valid Java Date format`	`DDD,MMM dd,yyyy`	The format to display a date with more information. Visit the website *http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html* to ensure the exact format.
timeFormat	`valid Java Date format`	`hh:mm a`	The format to view time (for example, hour, minute,). Visit the website: *http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html* to ensure the exact format.
maxTopic	`Integer`	10	The maximum number of topics per page.

Parameter	Possible value	Default value	Description
maxPost	`Integer`	10	The maximum number of posts per page.
isShowForumJump	`true / false`	`true`	Show the forum jump or not.

Answer Configuration

Answers Initializer

Overview

The Initialization plug-in component is to define the default answers data in the *.xml* or *.zip* file. It includes categories of question that should be exported from the Answers application.

When the Answers Service starts, it will get values returned from the Initialization plug-in component to initialize the default Answers data.

Configuration

- Default Answers data

The default Answers data is configured in the *services-configuration.xml* file.

In details, at runtime of Answers Service, the Initialization plug-in component is called, the *services-configuration.xml* file will be executed. The component-plugin named *addInitialDataPlugin* will refer to *org.exoplatform.faq.service.InitializeDataPlugin* to execute some objects to create default data.

The default data in the *.zip* file is initialized as follows:

```
<component-plugin>
  <name>technical-faq</name>
  <set-method>addInitialDataPlugin</set-method>
  <type>org.exoplatform.faq.service.InitialDataPlugin</type>
  <description>Initialize</description>
  <init-params>
    <value-param>
      <name>location</name>
      <description>location where Answers export format file is stored</description>
      <value>war:/data/Technical-FAQ.zip</value>
    </value-param>
  </init-params>
</component-plugin>
```

- In which,

Name	Set-Method	Type	Description
technical-faq	addInitialDataPlugin	org.exoplatform.faq.service.InitialDataPlugin	Initialize the data plugin.

- Init-param

Name	Possible value	Default value	Description
location	`string`	war:/data/ Technical-FAQ.zip	The location where the Answers export format file is stored.

If the default data is in the *XML* format:

```
<value>war:/data/Technical-FAQ.xml</value>
```

By default, the default Answers data can only import if the importing categories do not exist in database.

To initialize default data in multiple files, it is required to declare them in multiple plugins.

```
<component-plugin>

....

</component-plugin>
```

Answers Email Templates Configuration

Overview

Answers is configured mainly in the file:

- Portlet preferences: **/webapps/faq/WEB-INF/portlet.xml**

Note

For general information of eXo Knowledge configuration, refer to *eXo Knowledge Configuration* [KS:KS Manual#Configuration] section.

Configuration

The Mail templates use a specific syntax, enabling you to create a customized email message in the Edit mode via three templates: New question, Edit/answer, and Move question.

Parameters which are used in templates consist of:

Parameter	Description
&categoryName	Load the name of Category.
&questionContent	Load the question's content.
&questionResponse	Load the question's answer.
&questionLink	Load the link to question in the Answers portlet.

Poll Configuration

Overview

The Initialization plugin component defines the default Poll data in the *.xml* file, including polls. When the Poll Service runs, it will get values returned from the Initialization plugin component to initialize default Poll data.

Configuration

Default Poll data

The default Poll data are configured in the *war:webapp/WEB-INF/conf/ksdemo/ks/services-configuration.xml* file.

In particular, when the Poll service starts, the Initialization plug-in component is called. Next, the *services-configuration.xml* file is executed. The component-plugin named *addInitialDefaultDataPlugin* will refer to *org.exoplatform.poll.service.InitialDeafaultDataPlugin* to execute some objects to create default data for the Poll application.

```
<component-plugin>
  <name>default.data</name>
  <set-method>addInitialDefaultDataPlugin</set-method>
  <type>org.exoplatform.poll.service.InitialDefaultDataPlugin</type>
  <description>Initialize</description>
  <init-params>
    <object-param>
      <name>livedemo.default.configuration</name>
      <description>initial data for live demo</description>
      <object type="org.exoplatform.poll.service.PollInitialData">
```

```
      <field name="pollDatas">
        <collection type="java.util.ArrayList">
          <value>
            <object type="org.exoplatform.poll.service.PollData">
              <field name="parentPath">
                <string>ksdemo/Polls</string>
              </field>
              <field name="owner">
                <string>root</string>
              </field>
              <field name="question">
                <string>What color do you like ?</string>
              </field>
              <field name="timeOut">
                <string>0</string>
              </field>
              <field name="isMultiCheck">
                <string>false</string>
              </field>
              <field name="isAgainVote">
                <string>false</string>
              </field>
              <field name="isClosed">
                <string>false</string>
              </field>
               <field name="options">
               <collection type="java.util.ArrayList">
                <value><string>Green</string></value>
                <value><string>Blue</string></value>
                <value><string>Red</string></value>
                <value><string>Yellow</string></value>
                <value><string>Orange</string></value>
                <value><string>Purple</string></value>
               </collection>
              </field>
            </object>
          </value>
        </collection>
      </field>
    </object>
  </object-param>
 </init-params>
</component-plugin>
```

- In which,

Name	Set-method	Type	Description
default.data	`addInitialDefaultDataPlugin`	`org.exoplatform.poll.service.InitialDefaultDataPlugin`	The initial default data of Poll.

- Init-param

Name	Possible value	Default value	Description
livedemo.default.configuration	**configuration**	`org.exoplatform.poll.service.InitialData`	The initial data for live demo.

- Poll array

After the *org.exoplatform.poll.service.InitialDefaultDataPlugin* object has been executed, the *org.exoplatform.poll.service.PollInitialData* object will be called. It returns a polls array. The value of poll array is defined by the *org.exoplatform.poll.service.PollData* object as below:

```
<name>livedemo.default.configuration</name>
<description>initial data for live demo</description>
<object type="org.exoplatform.poll.service.PollInitialData">
 <field name="pollDatas">
  <collection type="java.util.ArrayList">
   <value>
    <object type="org.exoplatform.poll.service.PollData">
     ....
    </object>
   </value>
  </collection>
 </field>
</object>
```

- A Poll includes some basic data which are defined in the *field* tag with a specific name as below:

```
....
<field name="parentPath">
 <string>ksdemo/Polls</string>
</field>
<field name="owner">
 <string>root</string>
</field>
<field name="question">
```

```
  <string>What color do you like ?</string>
</field>
<field name="timeOut">
  <string>0</string>
</field>
<field name="isMultiCheck">
  <string>false</string>
</field>
<field name="isAgainVote">
  <string>false</string>
</field>
<field name="isClosed">
  <string>false</string>
</field>
<field name="options">
  <collection type="java.util.ArrayList">
    <value><string>Green</string></value>
    <value><string>Blue</string></value>
    <value><string>Red</string></value>
    <value><string>Yellow</string></value>
    <value><string>Orange</string></value>
    <value><string>Purple</string></value>
  </collection>
</field>
....
```

In which:

Field	Possible value	Default value	Description
parentPath	`string`	`ksdemo/Polls`	Parent path of Poll data.
owner	`user id`	`root`	The creator of Poll.
question	`string`	`What color do you like?`	The question for Poll.
timeout	`number`	`0`	The time before poll is closed. If value is set to *0*, the poll will never be closed.
isMultiCheck	`boolean`	`false`	If the value is *true*, user can vote for multi-options. If the value is *false*, only one option can be voted.

Field	Possible value	Default value	Description
isAgainVote	`boolean`	`false`	If the value is *true*, user can vote again.
isClose	`boolean`	`false`	If the value is *true*, the poll will be closed.
options	`java.util.ArrayList`	List of string	list of options for Poll.

- Modify values of Poll

Values of the default Poll can be changed by editing text values in the tag of each *field* by the other one.

Data Injector Servive

Data injector is used to create data for the performance benchmark. This part will describe which data injectors are implemented in eXo Knowledge and how to use them.

Technical details

In eXo Knowledge, data injectors are implemented as plugins attached to the org.exoplatform.services.bench.DataInjectorService service. This service is normally registered to the portal container as a general component and handled via RESTful requests.

To use this service, add the following dependency to the Classpath of the server:

```
<dependency>
  <groupId>org.exoplatform.commons</groupId>
  <artifactId>exo.platform.commons.component</artifactId>
  <version>1.1.3</version>
  <scope>provided</scope>
</dependency>
```

When you want to inject data for a specified product, you will have to implement a class which extends *org.exoplatform.services.bench.DataInjector* and register it to *DataInjectorService* as a plugin.

In which, methods need to be installed are:

```
public abstract class DataInjector extends BaseComponentPlugin {

  /**
```

```
  * get log object.
  * @return
  */
 public abstract Log getLog();

 /**
  * This function should be implemented to execute tasks that require to response data to client.
  * <br>
  * @param params query parameters of a HTTP GET request.
  * @return object that can be serialized to JSON object.
  * @throws Exception
  */
 public abstract Object execute(HashMap<String , String> params) throws Exception;

 /**
  * This function should be implemented to inject data into the product.
   * @param params parameters for injecting. They can be query parameters of a HTTP GET
request.
  * @throws Exception
  */
 public abstract void inject(HashMap<String , String> params) throws Exception;

 /**
  * This function should be implemented to clear data that is injected before by {@link #inject()}.
   * @param params parameters for rejecting. They can be query parameters of a HTTP GET
request.
  * @throws Exception
  */
 public abstract void reject(HashMap<String , String> params) throws Exception;
```

Configuration

To activate *DataInjectorService*, you must register this component to a portal container by the following configuration:

```
<component>
  <type>org.exoplatform.services.bench.DataInjectorService</type>
</component>
```

In eXo Knowledge, there are three plugins attached to the *DataInjectorService* component:

- ForumDataInjector

- WikiDataInjector
- AnswerDataInjector

ForumDataInjector

The Forum Data injector is configured to register to DataInjectorService by the following code:

```
<external-component-plug-ins>
  <target-component>org.exoplatform.services.bench.DataInjectorService</target-component>
  <component-plug-in>
    <name>ForumDataInjector</name>
    <set-method>addInjector</set-method>
    <type>org.exoplatform.forum.bench.ForumDataInjector</type>
    <description>inject data for Forum</description>
  </component-plug-in>
</external-component-plug-ins>
```

- To inject data, the request link is in the following format:

```
http://[rest-path]/bench/inject/ForumDataInjector/
?type=data&q=10,20,30,40,50&pre=cat,for,top,post,att&attSize=1
```

The query parameters description:

Param	Value	Description
q	Number	The quantity of each item in forum. For example, if the value is 10,20,30,40,50 then the injector will create 10 categories, each category will have 20 forums, each forum contains 30 topics, each topic has 40 posts which contains 50 attachments in each posts.
pre	String	The prefixes of category, forum, topic, post and attachment. If "cat,for,top,pos,att" is inputed, the injector will create a set of data inlcude: categories with

Param	Value	Description
		"cat" prefix, forums with "for" prefix and so on.
attSize	Number	The size of each attachment which is created in a post.

- Every created topic can only be read/modified by user 'root' to grant permission to other members.

Wiki Data injector

The Data injector for Wiki is configured as follow:

```
<external-component-plug-ins>
  <target-component>org.exoplatform.services.bench.DataInjectorService</target-component>
  <component-plug-in>
    <name>WikiDataInjector</name>
    <set-method>addInjector</set-method>
    <type>org.exoplatform.wiki.bench.WikiDataInjector</type>
    <description>inject data for Wiki</description>
  </component-plug-in>
</external-component-plug-ins>
```

- To inject data, the request link is in the following format:

```
http://[rest path]/bench/inject/WikiDataInjector/
?type=data&q=1,2,3&pre=abc,def,mnp&wo=classic&wt=portal&maxAtt=10&mP=100
```

In which, parameters mean:

Param	Value	Description
type	[data | perm]	Type of injector. It can be data or perm which injects data or permission respectively.
q	Number	The number of pages in each depth, separated by commas. For example, if value is 1,2,3 then the injector will create 1 new child of WikiHome, 2 children per each page

Param	Value	Description
		created in depth 1 and 3 children for each page created in depth 2.
pre	String	Prefix for page id in each depth, separated by commas. For example, if value is abc,def,ghk then pages in depth 1 have title starting with "abc", title of pages in depth 2 start with "def" and in depth 3 is "ghk".
wo	String	Wiki owner.
wt	String	Wiki type. The value can be: 'portal', 'user', 'group'.
maxAtt	Number	The size of attachment in created pages. the value is evaluated in KByte. If the value is 0 or not set, no attachment is created.
mP	Number	The maximum pages in each injection. The number of created pages must not exceed this value.

To grant permision, the request link is in the following format:

```
http://localhost:8080/rest-ksdemo/private/bench/inject/WikiDataInjector/
?type=perm&q=1,2,3&pre=abc,def,ghk&wo=classic&wt=portal&perm=11&users=root,mary&groups=*:/
platform/user&rcs=true
```

For instance, in the link above, the injector will set the Read and Edit permission for pages of portal/classic which meet the constraint (q=1,2,3 and pre=abc,def,ghk).

You can use these parameters to set up permissions for pages:

Param	Value	Description
q	Number	The number of pages in each depth separated by commas. For example, if value is 1,2,3 then the injector will create

Param	Value	Description
		1 new child of WikiHome, 2 children per each page created in depth 1 and 3 children for each page created in depth 2.
pre	String	The prefix for page id in each depth separated by commas. For example, if the value is "abc,def,ghk" then pages in depth 1 will have title starting with "abc", title of pages in depth 2 will start with "def" and in depth 3 is "ghk"
wo	String	The wiki owner separated by commas.
wt	String	The wiki type separated by commas. This value can be: portal, user or group.
users	String,	The list of granted permissions users separated by commas.
groups	String	The list of granted permissions groups separated by commas.
memship	String	The list of granted permissions memberships separated by commas.
perm	*number number*	The declared permissions. The value must be a string with 2 numbers. The first number is to define Read permission of identity while the other one is for Edit permission. If the number is "zero", the privilege is denied and vise versa. For example, 11 means that both Read and Edit pages permmission are granted.
rcs	Boolean	Recursive or not. If the value is true, all pages that meet the constraint will be set permission, or deepest pages

Param	Value	Description
		(smallest descendants) will be affected.

AnswerDataInjector

The following configuration is used for the Answer Data injector:

```
<external-component-plug-ins>
  <target-component>org.exoplatform.services.bench.DataInjectorService</target-component>
  <component-plug-in>
    <name>AnswerDataInjector</name>
    <set-method>addInjector</set-method>
    <type>org.exoplatform.faq.bench.AnswerDataInjector</type>
    <description>inject data for Answer</description>
  </component-plug-in>
</external-component-plug-ins>
```

- To inject data, the request link is in the following format:

```
http://[rest                                    path]/bench/inject/AnswerDataInjector/
?type=data&q=2,3,4,5,6&pre=cate,ques,answ,comm&att=2&attCp=100&&txtCp=100
```

In which:

Param	Value	Description	
type	*data* [perm]	Type of injector. It can be data or perm means injecting data or permission respectively.	
q	number,number,etc	The number of items in each depth. For example, if the value is 2,3,4,5,6 then the injector will create 2 new categories with the depth level is '3', add 4 questions	

Param	Value	Description	
		in each category, 5 answer and 6 comment in each question. **Warning**: Do not set the depth level to more than 5, because the number of items are calculated by number*depth.	
pre	string, string, etc	The prefix for items id in each depth. For example, if the value is "cate, ques, answ, comm" then category has fist name/id is "cate", question has fist name/id is "ques", answer has fist id is "answ" and comment has fist id is "comm" .	
att	number	The number of attachments in one question. If the value is '0' or not set, no attachment is created.	
attCp	number	The capacity of one attachment. The value is evaluated in KByte and must be larger than KByte.	
txtcp	number	The capacity text of one item (question/ answer/comment). The value is evaluated in KByte. If the value is 0 or not set, texts are created randomly.	

Note

All number of item injectors are calculated by:

- Categories: cats = numberCat * depth!

- Questions : ques = cats * numberQue
- Answers : anss = quest * numberAns
- Comments : coms = quest * numberComs

All = cats + ques + anss + coms

For example:

q=2,3,4,5,6: All = 2 * 3! + (2 * 3) * 4) * 5 + ((2 * 3!) * 4) * 6 = 588 (items)

If you change the depth from '3' to '5', the number of items will be 11760.

- To grant permision, the request link is in the following format:

```
http://[rest path]/bench/inject/AnswerDataInjector/
?type=perm&q=2,3,4,5,6&pre=cate,ques,answ,comm&att=2&attCp=100&&txtCp=100&view=root,demo,*:/
platform/user&edit=root,*:/platform/manager
```

In which:

Param	Value	Description	
view	string, string, etc	The list of granted permissions users/ groups/memberships, if the value is "any" or not set, everyone can view all category create by prefix.	
edit	string, string, etc	The list of granted permissions users/ groups/memberships, if the value is "any" or not set, only user with highest permission (Administrator) can edit all category create by prefix.	

Note

In the other way, such settings have been declared in "ksdemo.war/WEB-INF/conf/ksdemo/ks/bench-configuration.xml", therefore, to save time and effort, you can import

it to "ksdemo.war/WEB-INF/conf/configuration.xml" and then modify it rather than create new one.

How to use?

You can use RESTful service to request to inject or reject data. The format of request link is:

```
http://{domain}/{rest}/bench/{inject|reject}/{plug-inName}?[params]
```

For example, after registering the WikiDataInjector plug-in as above, you can request injection as follows: *http://localhost:8080/rest-ksdemo/bench/inject/WikiDataInjector?mP=10&mA=100&mD=1&rand=false&wo=classic&wt=portal* with 10 childrens of each page created, 100 kb each attachment, 1 depth level for wiki portal classic.

To reject such created data, request this link: *http://localhost:8080/rest-ksdemo/bench/reject/WikiDataInjector.*

JCR structure

Overview

eXo Knowledge is a JCR-based product, it uses JCR to store data. The root node of eXo Knowledge is exo:application which includes these child nodes: exo:forumHome, exo:faqHome.

Forum JCR structure

Forum is a JCR-based application. The Forum data are saved in eXo-JCR under the Forum Service data directory. The whole JCR structure of Forum can be visualized in the diagram below:

Forum System

The Forum System node is created from the node type exo:forumSystem. That is defined as a child node of Forum Service and can store nodes with these following node types: exo:banIP, exo:forumUserProfile, exo:statistic, exo:administration under the Forum System. The Forum System node is stored in `/exo:applications/ForumService/ForumSystem`.

User Profile and User Profile Home

The User Profile and User Profile Home node are used to store information of each user. User Profile is automatically created by a listener when a user registers to the organization service. Private message and forum subscription can be added to User Profile as a child node. These node types exo:forumUserProfile, exo:userProfileHome, exo:privateMessage and exo:forumSubscription are defined as child nodes of exo:forumUserProfile. The User Profile node is stored under ForumSystem node: `/exo:applications/ForumService/ForumSystem/exo:userProfileHome/exo:forumUserProfile`.

- The node type exo:forumUserProfile has the following properties:

Properties name	Required type	Multiple	Description
exo:userId	String	false	The user id.
exo:fullName	String	false	The user's full name.
exo:firstName	String	false	The user's first name.
exo:lastName	String	false	The user's last name.
exo:email	String	false	The user's email address.
exo:userTitle	String	false	The user's title: Administrator, Moderator or User.
exo:screenName	String	false	

Properties name	Required type	Multiple	Description
			The displayed name of user in Forum.
exo:userRole	Long	false	The user's role. The value can be: "O": Administrator, "1": Moderator, "2": User, "3": guest.
exo:signature	String	false	The signature displayed at the end of each user's post.
exo:totalPost	Long	false	The total posts submitted by the user.
exo:totalTopic	Long	false	The total topics started by the user.
exo:jobWattingForModeration	Long	false	The number of jobs that are waiting to be moderated.
exo:moderateForums	String	true	The list of forum ids that user is the moderator.
exo:moderateCategory	String	true	The list of category ids that user is the moderator.
exo:readTopic	String	true	The list of topics that user has read.
exo:readForum	String	true	The list of forums that user has read.
exo:lastReadPostOfTopic	String	true	The list of the last read posts id in a topic that user has read.
exo:lastReadPostOfForum	String	true	The list of the last read posts id in a forum that user has read.
exo:isAutoWatchMyTopics	Boolean	false	Enable/disable the auto-watch the topics created by user. Topics created by a user will be watched

Properties name	Required type	Multiple	Description
			automatically if the value is set to "true".
exo:isAutoWatchTopicIPost	Boolean	false	Enable/disable the auto-watch posts submitted by user. Topics posted by an user will be watched automatically if the value is set to "true".
exo:bookmark	String	true	The list of topics/posts bookmarked by user.
exo:lastLoginDate	Date	false	The date of the last login.
exo:joinedDate	Date	false	The date when user joined forum.
exo:lastPostDate	Date	false	The date of the last post.
exo:isDisplaySignature	Boolean	false	User's signature will be displayed at the end of their post if the value is set to "true".
exo:isDisplayAvatar	Boolean	false	User's avatar is displayed if the value is set to "true".
exo:newMessage	Long	false	The number of new messages.
exo:timeZone	Double	false	The time zone configured by user.
exo:timeFormat	String	false	The time format configured by user: 12h or 24h format.
exo:shortDateformat	String	false	The format of short date configured by user. Example: 'dd/MM/yyyy'.
exo:longDateformat	String	false	The format of long date configured by user. Example: 'dd mmm, yyyy'.

Properties name	Required type	Multiple	Description
exo:maxPost	Long	false	The number of the maximum posts displayed per page.
exo:maxTopic	Long	false	The number of the maximum topics displayed per page
exo:isShowForumJump	Boolean	false	Display/hide the forum jump drop-down list. This jump list will be shown if the value is set to "true".
exo:collapCategories	String	true	The list of categories collapsed by user.
exo:isBanned	Boolean	false	The user's condition. User is currently banned if the value is set to "true".
exo:banUntil	Long	false	The time when the ban period expires.
exo:banReason	String	false	The description for the reason that user was banned.
exo:banCounter	String	false	The number of bans that user has committed.
exo:banReasonSummary	String	true	The list of ban reason summaries when a user is banned for more than one time.
exo:createdDateBan	Date	false	The date when the ban period starts.

- The child node type exo:privateMessage has the following properties:

Properties name	Required type	Multiple	Description
exo:from	String	false	The user id of the sender.
exo:sendTo	String	false	The user id of the receiver.

Properties name	Required type	Multiple	Description
exo:name	String	false	The private message subject.
exo:message	String	false	The message contents.
exo:type	String	false	The private message type: sent messages or received messages.
exo:receivedDate	Date	false	The date when the private message was received.
exo:isUnread	Boolean	false	The status of private message: read/unread.

- The child node type exo:forumSubscription has the following properties:

Properties name	Required type	Multiple	Description
exo:categoryIds	String	true	The ids of the subscribed categories.
exo:forumIds	String	true	The ids of the subscribed forums.
exo:topicIds	String	true	The ids of the subscribed topics.

Statistic and Statistic Home

Statistic and Statistic Home are used to store statistic information of forum, such as number of posts, topics, users, active users. The node type is exo:forumStatistic, exo:statisticHome.

- The Statistic node is stored under the Forum System node: `/exo:applications/ForumService/ForumSystem/exo:statisticHome/exo:forumStatistic` and its node type has the following properties:

Properties name	Required type	Multiple	Description
exo:postCount	Long	false	The total number of submitted posts in Forum.
exo:topicCount	Long	false	The number of total created topics in Forum.

Properties name	Required type	Multiple	Description
exo:membersCount	Long	false	The number of the registered users.
exo:newMembers	String	false	The id of the latest registered user.
exo:mostUsersOnline	String	false	The highest number of the online users.
exo:activeUsers	Long	false	The number of active users.

Ban IP and Ban IP Home

The Ban IP and Ban IP Home node are used to store data about banned IP addresses. The node type exo:banIPHome contains the child node exo:IPHome.

- The Ban IP node is stored under the Forum System node: `/exo:applications/ForumService/ForumSystem/exo:banIPHome/exo:banIP` and its node type has the following properties:

Properties name	Required type	Multiple	Description
exo:ips	String	true	The list of ip addresses of the banned users.

Administration and Administration Home

Administration and Administration Home are used to store data for setting the layout, notification email format and censor jobs. The node type of the Administration Home node is exo:administrationHome and the its child node type is exo:administration. The Administration node is stored under the ForumSystem node `/exo:applications/ForumService/ForumSystem/exo:administrationHome/exo:administration` and its node type has the following properties:

Properties name	Required type	Multiple	Description
exo:forumSortBy	String	false	Sort forum by criteria: post count, topic count, lock status.
exo:forumSortByType	String	false	Sort forum by ascending/ descending type.
exo:topicSortBy	String	false	Sort topic by criteria.
exo:topicSortByType	String	false	Sort topic by ascending type or descending type.

Properties name	Required type	Multiple	Description
exo:censoredKeyword	String	false	The list of censored words.
exo:headerSubject	String	false	The subject header.
exo:enableHeaderSubject	Boolean	false	Enable/disable the subject header. The subject header is displayed if the value is set to "true".
exo:notifyEmailContent	String	false	Define if the notification email will be sent when there is a new added topic/post.
exo:notifyEmailMoved	String	false	Define if the notification email will be sent when there are any moved topic/post.

Forum Data

The Forum Data node is created from the node type exo:forumData. The data nodes like category, forum, topic, post, tag, BBcode and topic type will be stored under the Forum Data node:`/exo:applications/ForumService/ForumData`.

Category and Category home

The Category node is used to store all categories of forum, this node is a child node of the Forum Data node and only Category node type can be added to the Category Home. The node type of the Category Home node is exo:categoryHome is stored in `/exo:applications/ForumService/ForumData/CategoryHome`. The Category node has the node type exo:forumCategory which is a child node of the CategoryHome node. This node type is defined to allow adding child nodes as exo:forum and exo:forumRSS.

- The node type exo:forumCategory has the following properties:

Properties name	Required type	Multiple	Description
exo:id	String	false	The category id.
exo:owner	String	false	The category creator.
exo:path	String	false	The node path of the category.
exo:createdDate	Date	false	

Properties name	Required type	Multiple	Description
			The date when the category was created.
exo:modifiedBy	String	false	The id of the user who made the last modification on the category.
exo:name	String	false	The category name.
exo:modifiedDate	Date	false	The date when the modifications on category were made.
exo:description	String	false	The category description.
exo:moderators	String	true	The list of moderators of the category.
exo:tempModerators	String	true	The temporary moderator of the category.
exo:createTopicRole	String	true	The topic role.
exo:poster	String	true	The list of ids of the users and groups who can post in the category.
exo:viewer	String	true	The list of ids of the users and groups who can only view posts in the category.
exo:categoryOrder	Long	false	The order number of category in the category list.
exo:userPrivate	String	true	The list of user ids whose access are restricted from the category.
exo:forumCount	Long	false	The total number of forums in the category.

Forum

The Forum node is defined as a child node of category and allowed adding child nodes as Topic and RSS type. The node type of Forum is exo:forum. The Forum node is stored in `/exo:applications/ForumService/ForumData/CategoryHome/%Category-id%/%Forum-id%` and its node type has the following properties:

Properties name	Required type	Multiple	Description
exo:id	String	false	The forum id.
exo:owner	String	false	The forum creator.
exo:path	String	false	The node path of the forum.
exo:name	String	false	The forum title.
exo:forumOrder	Integer	false	The order number in the list of forums. Forum with smaller number will get higher order.
exo:createdDate	Date	false	The date and time when the forum was created.
exo:modifiedBy	String	false	The id of user who modified the category.
exo:modifiedDate	Date	false	The time of modification, including date, time and time zone.
exo:lastTopicPath	String	false	The id of the last topic in the forum.
exo:description	String	false	The description of forum.
exo:postCount	Long	false	The total number of submitted posts in the forum.
exo:topicCount	Long	false	The total number of created topics in the forum.
exo:isAutoAddEmailNotify	Boolean	false	Enable/disable the notification email to moderators.
exo:notifyWhenAddTopic	String	true	

Properties name	Required type	Multiple	Description
			Email addresses to notify when there is a new topic in the forum.
exo:notifyWhenAddPost	String	true	Email addresses to notify when there is a new post in the forum.
exo:isModerateTopic	Boolean	false	All new topic will be moderated if the value is set to "true".
exo:isModeratePost	Boolean	false	All new posts will be moderated if the value is set to "true".
exo:isClosed	Boolean	false	The forum status: closed/open. Forum is closed if the value is set to "true".
exo:isLock	Boolean	false	The forum status: locked/unlocked. Forum is locked if the value is set to "true".
exo:createTopicRole	String	true	The list of ids of the users or groups who can create topic in the forum.
exo:poster	String	true	The list of ids of the users or groups who can submit post in the forum.
exo:viewer	String	true	The list ids of the users or groups who can view posts in the forum.
exo:moderators	String	true	The list of user ids who are the moderators of forum.
exo:tempModerators	String	true	The list of user ids who are the temporary moderators.
exo:banIPs	String	true	The list of banned IP addresses.

- The child node pruneSetting has the following perperties:

Properties name	Required type	Multiple	Description
exo:id	String	False	The forum id.
exo:inActiveDay	Long	False	The number of days/ weeks/months that the topics in forum have not been active.
exo:periodTime	Long	False	The number of days/ weeks/months that the prune job will be executed to check for the old topics and deactivate them.
exo:isActive	Boolean	False	The current status of the prune job. If the value is set to "True", the prune job will be run.
exo:lastRunDate	Date	False	The date that prune job runs for the last time.

Topic

The Topic node is defined as a child node of the Forum node and allowed adding child nodes as Topic, Poll and RSS type. The node type of the Topic and Poll node is exo:topic, exo:poll.

- The Topic node is stored in `/exo:applications/ForumService/ForumData/CategoryHome/%Category-id%/%Forum-id%/%Topic-id%` and its node type has the following properties:

Properties name	Required type	Multiple	Description
exo:id	String	false	The topic id.
exo:owner	String	false	The user id of the topic creator.
exo:path	String	false	The node path of the topic.
exo:name	String	false	The subject of the topic
exo:createdDate	Date	false	The time when the topic was created.
exo:modifiedBy	String	false	The id of the user who made the latest

Properties name	Required type	Multiple	Description
			modification in the topic.
exo:modifiedDate	Date	false	The date when the modifications were made.
exo:lastPostBy	String	false	The user id of the last poster in topic.
exo:lastPostDate	Date	false	The date when the last post was submitted.
exo:description	String	false	The topic description.
exo:topicType	String	false	The id of the topic type.
exo:postCount	Long	false	The number of posts in the topic.
exo:viewCount	Long	false	The number of topic views.
exo:numberAttachments	Long	false	The number of attachments in the topic.
exo:icon	String	false	The name of the topic icon.
exo:link	String	false	The link to the topic. Example: http://localhost:8080/ksdemo/public/classic/forum/topic/%Topic-id%.
exo:isModeratePost	Boolean	false	All posts in the topic will have to wait for moderation if the value is set to "true".
exo:isNotifyWhenAddPost	Boolean	false	When there is a new post in a topic, a notification message will be sent to the topic owner if this value is set to "true".
exo:isClosed	Boolean	false	The state of the topic: closed/open. If the

Properties name	Required type	Multiple	Description
			value is set to "true", the topic is closed.
exo:isLock	Boolean	false	The lock status of the topic: lock/unlocked. If the value is set to "true", the topic is locked.
exo:isApproved	Boolean	false	The topic is approved to be published if the value is set to "true".
exo:isSticky	Boolean	false	If the value is set to "true", the topic is currently sticky.
exo:isWaiting	boolean	false	The topic status. The topic is waiting for moderation if the value is set to "true".
exo:isActive	boolean	false	The topic activity status: active/inactive. The topic is active (topic gets new posts in a period of time) if the status is set to "true".
exo:isActiveByForum	Boolean	false	The topic status based on the forum state. Example: When the topic is active and the forum that contains it is closed, this topic will be considered as inactive.
exo:canView	String	true	List of user ids who can view the topic contents.
exo:canPost	String	true	List of user ids who can post in the topic.
exo:isPoll	Boolean	false	The topic contains poll if the value is set to "true".

Properties name	Required type	Multiple	Description
exo:userVoteRating	String	true	The list of user id who voted.
exo:tagId	String	true	The list of the topic tag id.
exo:voteRating	Double	false	The average vote score of the topic.

- The child node type exo:poll has the following properties:

Properties name	Required type	Multiple	Description
exo:id	String	false	The poll id.
exo:owner	String	false	The user id of poll creator.
exo:createdDate	Date	false	The date and time when the poll was created.
exo:modifiedBy	String	false	The user id who modified the poll.
exo:modifiedDate	Date	false	The time when the poll is modified.
exo:lastVote	Date	false	The date of the last vote.
exo:question	String	false	The contents of the question for poll.
exo:timeOut	Long	false	The time when the polled is closed.
exo:option	String	true	The list of options for poll.
exo:vote	String	true	The list of votes by users.
exo:userVote	String	true	The list of user ids who voted.
exo:isMultiCheck	Boolean	false	User can choose more than one option if the value is set to "true".
exo:isAgainVote	Boolean	false	Users can change their vote if the value is set to "true".

Properties name	Required type	Multiple	Description
exo:isClosed	Boolean	false	The poll status. Poll is closed if the value is set to "true".

Post

The Post node is defined as child node of Topic and allowed adding only the Attachment child node type. The Post node type is exo:post, and the child node type is exo:forumAttachment.

- The Post node is stored in `/exo:applications/ForumService/ForumData/CategoryHome/%Category-id%/%Forum-id%/%Topic-id%/%Post-id%` and its node type has the following properties:

Properties name	Required type	Multiple	Description
exo:id	String	false	The post id.
exo:owner	String	false	The user id of the poster.
exo:path	String	false	The node path of the post.
exo:createdDate	Date	false	The date time when post is submitted, including date, time, time zone.
exo:modifiedBy	String	false	The id of the user who modified the post.
exo:editReason	String	false	The reason for editing the post.
exo:modifiedDate	Date	false	The date when the post was modified.
exo:name	String	false	The post title.
exo:message	String	false	The message of the post.
exo:remoteAddr	String	false	The remote IP address of the post.
exo:icon	String	false	The name of the icon for the post.
exo:userPrivate	String	true	The list of user ids that are restricted from the post.

Properties name	Required type	Multiple	Description
exo:link	String	false	The link to open the topic.
exo:isApproved	Boolean	false	The state of the post: approved/ unapproved. The post is approved if the value is set to "true".
exo:numberAttach	Long	false	The number of attachments in the post.
exo:isActiveByTopic	Boolean	false	The post is activity status based on the topic state. If the topic is close, all post in it will be considered as inactive.
exo:isHidden	Boolean	false	The post status: shown/hidden. The post is hidden if the value is set to "true".
exo:isFirstPost	Boolean	false	The post is the first one in a topic if the value is set to "true".

Tag and Tag home

The Tag node is used to store data about tag name, topics with tag added, number of users using this tag, number of tags in use. The node type of the Tag node is exo:forumTag and its child node type is "exo:tagHome". The Tag node is stored in `/exo:applications/ForumService/ForumData/TagHome/%tag-id%` and its node type has the following properties:

Properties name	Required type	Multiple	Description
exo:id	String	false	The id of tag.
exo:name	String	false	The tag name.
exo:useCount	Long	false	The number of times that the tag was used.
exo:userTag	String	true	The number of users using the tag.

BBCode and BBCode home

The BBCode node is used to define what BBCode will be used in the forum. The node type of the BBCode node is exo:forumBBCode. The BBCode node is stored in `/exo:applications/ForumService/ForumData/forumBBCode/%BBCode_tag%` and its node type has the following properties:

Properties name	Required type	Multiple	Description
exo:description	String	false	The description about the tag purpose. Example: 'The [url] tag allows creating links to other websites and files'.
exo:example	String	false	The example about using the tag. Example: [URL]http://www.exoplatform.com[/URL]'.
exo:isActive	Boolean	false	The BBCode tag is active/deactive. The BBCode tag is active if the value is set to "true".
exo:isOption	Boolean	false	If the value is set to "true", user can create a tag with atributes and values.
exo:replacement	String	false	The HTML code that will be replaced by the tag. Example: [url] tag replaces '{param}'.
exo:tagName	String	false	The BBCode tag name.

Topic type and Topic type home

The Topic type home contains a child node with the node type exo:topicType. The Topic node is stored in `/exo:applications/ForumService/ForumData/TopicTypeHome/%Topic-id%` and its node type has the following properties:

Properties name	Required type	Multiple	Description
exo:id	String	false	The id of the topic type.
exo:name	String	false	The name of the topic type.
exo:icon	String	false	The icon of the topic type.

FAQ JCR structure

FAQ is a JCR-based application. The FAQ data are stored in the eXo-JCR under the faqApp data directory. The whole FAQ JCR structure can be visualized in the following diagram:

Category

The Category node is created from the exo:faqCategory node type and defined to add data inside as:

- Sub-category
- Question Home
- RSS

Sub-category

The system will automatically create the Category Home node under the FAQ application node at the first time the user launches application. All users-created categories are the sub-categories of Category Home. The home of the Category node is automatically created in `/exo:applications/faqApp/categories`.

In fact, Sub-category is also a category. FAQ has defined a mixin node type called mix:faqSubCategory to allow adding a node having the same type with category to an existing category. When a category is created, this mixin node type will be mixed to that category.

The node type exo:faqCategory has the following properties:

Properties name	Required type	Multiple	Description
exo:id	String	false	The sub-category id.
exo:name	String	false	The name of the sub-category.
exo:userPrivate	String	true	The list of user ids that are restricted from the category.

Properties name	Required type	Multiple	Description
exo:description	String	false	The description of the sub-category.
exo:isModerateQuestions	Boolean	false	The question post moderation status. All questions posted in the sub-category will have wait for moderation if the value is set to "true".
exo:isModerateAnswers	Boolean	false	The answer post moderation status. All answers posted in the sub-category will have to wait for moderation if the value is set to "true".
exo:isView	Boolean	false	The category is shown/hidden. The category will be shown if the value is set to "true".
exo:viewAuthorInfor	Boolean	false	The category enables user to view the information of questions poster if the value is set to "true'.
exo:moderators	String	true	The list of user ids who are the category moderator.
exo:createdDate	Date	false	The time when the sub-category is created.
exo:index	Long	false	The index number of the category.

RSS

Each category has a RSS child node that stores a RSS feed representing all questions in this category as the binary data type. The RSS node is stored in `/exo:applications/faqApp/categories/ks.rss` and its node type is exo:faqRSS.

Properties name	Required type	Multiple	Description
exo:content	Binary	false	The content of the RSS.

Question and Question Home

The Question Home node is created from the exo:faqQuestionHome node type that is defined as a child node of category. This node cantains all question nodes that created in side a category. Only the Question node type exo:faqQuestion can be added to the question Home. The Question Home node is created as a child node of Categories `/exo:applications/faqApp/categories/questions`.

Question node is created from exo:faqQuestion node type under the Question Home node. The Answers, Comments and Attachments node are defined as child nodes of the Question node. The Question node is created under the Question Home: `/exo:applications/faqApp/categories/questions/%Question-id%`.

- The exo:faqQuestion node type has the following properties:

Properties name	Required type	Multiple	Description
exo:id	String	false	The question id.
exo:language	String	false	The language of the question.
exo:name	String	false	The question details.
exo:title	String	false	The question title.
exo:author	String	false	The the user id of the question poster.
exo:email	String	false	The email of the question author
exo:isActivated	Boolean	false	The question status: activated/inactivated. The question is activated if the value is set to "true".
exo:isApproved	Boolean	false	The state of the question: approved/ unapproved. The question is approved to be published if the value is set to "true".
exo:categoryId	String	false	The id of the category containing the question.

Properties name	Required type	Multiple	Description
exo:createdDate	Date	false	The date and time when the question was submitted.
exo:relatives	String	true	The list of the related questions ids.
exo:usersVote	String	true	The list of user ids who voted.
exo:markVote	Double	false	The average vote scores of the question.
exo:topicIdDiscuss	String	false	The topic id in the forum where the question is discussed.
exo:nameAttachs	String	true	The file name of attachments in the question.
exo:lastActivity	String	false	The user id and time when the last activity of the question was made.
exo:numberOfPublicAnswers	Long	false	The number of all posted answers that has been published.
exo:link	String	false	The link to open the question.
exo:responses	String	true	The responses of the question.
exo:dateResponse	Date	true	The date when the question received the answer.
exo:responseBy	String	true	The user id of the answer poster.

Multilanguages

A question can support multilanguages, all other languages are stored as a child node of the question and can be add to the question via a mixin node type called mix:faqi18n. After the mixin node type mix:faqi18n is added to the question, the node type exo:questionLanguageHome can be added to the question node and this node type will contain all languages node with the node type exo:faqLanguage. All display properties of the question are defined in the node type exo:faqLanguage.

- The node type exo:faqLanguage has the following properties:

Properties name	Required type	Multiple	Description
exo:language	String	false	The language of the question.
exo:name	String	false	The name of the language.
exo:title	String	false	The title of the question in the selected language.
exo:questionId	String	false	The id of the question.
exo:categoryId	String	false	The id of the category.

Answer, Comment and Attachment

The Answer, Comment and Attachment node is defined as the child nodes of the Question node. Attachment node is defined as a nt:file node type and stored right under the Question node. Answers and comments node are stored under the Answer home and the Comment home node.

- The Answer node is stored in `/exo:applications/faqApp/categories/questions/%Question-id%/faqAnswerHome/%Answer-id%` and its node type has the following properties:

Properties name	Required type	Multiple	Description
exo:id	String	false	The id of the answer.
exo:answerPath	String	false	The path to the answer.
exo:questionId	String	false	The id of the question.
exo:categoryId	String	false	The id of the category containing the question.
exo:responses	String	false	The content of the answer.
exo:dateResponse	Date	false	The date when the response was posted.
exo:responseBy	String	false	The id of the user who responded the answer.
exo:responseLanguage	String	false	The language of the answer response.
exo:approveResponses	Boolean	false	The response is pending for approval

Properties name	Required type	Multiple	Description
			if the value is set to "false".
exo:activateResponses	Boolean	false	The state of the answer: activated/ deactivated .
exo:usersVoteAnswer	String	true	The list of user ids who voted for the answer.
exo:MarkVotes	Long	false	The average vote scores of the answer.
exo:postId	String	false	The post id.
exo:fullName	String	false	The answer author's full name.

- The Comment node is stored in `/exo:applications/faqApp/categories/questions/%Question-id%/faqCommentHome/%Comment-id%` and its node type has the following propeties:

Properties name	Required type	Multiple	Description
exo:id	String	false	The comment id.
exo:comments	String	false	The comment contents.
exo:dateComment	Date	false	The date when the comment is posted.
exo:commentBy	String	false	The user id of the comment poster.
exo:postId	String	false	The id of the post.
exo:fullName	String	false	The full name of the comment poster.
exo:categoryId	String	false	The id of the category in which the comment is posted.
exo:questionId	String	false	The id of the question in which the comment is posted.
exo:commentLanguage	String	false	The language of the comment.

- The Attachment node is stored in `/exo:applications/faqApp/categories/questions/%Question-id%/faqAttachment` and its node type has the following properties:

Properties name	Required type	Multiple	Description
exo:fileName	String	false	The name of the attachment file.

FAQ setting

This FAQ Setting node stores the user settings data, such as how answer is ordered (in alphabetical order or by created date), the order type (descending or ascending) or the user's selection to sort questions by popularity. Each user has a dedicated settings data to select the display preferences in FAQ . The default setting will be used if the users has never changed and saved their setting.

- The User Setting node of an individual user is stored in `/exo:applications/faqApp/settingHome/userSettingHome/%user-id%` and has the following properties:

Properties name	Required type	Multiple	Description
exo:ordeBy	string	false	Define how questions are ordered, by "alphabet/index" or "created date".
exo:ordeType	string	false	The value "asc" = ascending and "des" = descending.
exo:sortQuestionByVote	Boolean	false	All questions will be sorted by the popularity (based on the number of votes) if the value is set to "true".

Template for FAQ

This node stores the template for FAQ portlet. The user can edit this template online in FAQ to change the layout, skins, and more.

- The template is stored in an nt:file node type under the Template Home node.

`/exo:applications/faqApp/templateHome/nt:file`.

Poll JCR structure

The Poll data are saved in eXo-JCR under the eXoPolls data directory. The whole JCR structure of Poll can be visualized in the diagram below:

The Poll node is used to store the default data in a poll. The node type of the Poll node is exo:poll. The Poll node is stored under eXoPolls node `/exo:applications/eXoPolls/%PortalName%/Polls/Poll-id%` and its node type (exo:polls) has the following properties:

Properties name	Required type	Multiple	Description
exo:id	String	false	The poll id.
exo:owner	String	false	The user id of the poll creator.
exo:createdDate	Date	false	The date and time when the poll is created.
exo:modifiedBy	String	false	The id of the user who made the last modification on the poll.
exo:modifiedDate	Date	false	The date and time when the latest modification on poll was made.
exo:lastVote	Date	false	The date and time when the last vote was made.
exo:question	String	false	The question content of poll.
exo:timeOut	Long	false	The time when the poll will be closed.
exo:option	String	true	The list of options for poll. Each option is separated by a comma.
exo:vote	String	true	The list of votes by users.
exo:userVote	String	true	The list of user ids who voted.
exo:isMultiCheck	Boolean	false	Enable/disable the multi check. User can vote for more than one option if the value is set to "true".
exo:isAgainVote	Boolean	false	Enable/disable the option to vote again.

Properties name	Required type	Multiple	Description
			User can change their vote if the value is set to "true".
exo:isClosed	Boolean	false	The poll status: open/ closed. The poll is closed if the value is set to "true".

Wiki JCR structure

The Wiki portlet is a JCR-based application. The Wiki data are stored in the eXo JCR storage. The Wiki application of eXo Knowledge supports several wikis, each wiki is organized as a tree of pages and hosted in different locations, depending on the type of wiki:

- Portal wikis: */exo:applications/eXoWiki/wikis/$PORTAL/WikiHome*
- Group wikis: */Groups/$GROUP/ApplicationData/eXoWiki/WikiHome*
- User wikis: */Users/$USERNAME/ApplicationData/eXoWiki/WikiHome*

WikiHome a conventional name of the root page of a Wiki. The type of *WikiHome* is "exo:wikihome". Each page may have a number of sub-pages and attachments.

Other Wiki metadata are organized below */exo:applications/eXoWiki/wikimetadata*.

The whole Wiki JCR structure can be visualized in the following diagram:

Wiki data

Depending on the Wiki type (portal, group or user); its pages, attachments and default syntax are stored under a node with either of node types: "wiki:portalwiki", "wiki:groupwiki" or "wiki:userwiki", respectively. Such a node has four child nodes: WikiHome, Preferences, LinkRegistry and Trash. These node types have the super type "wiki:wiki" that has the following properties:

Properties name	Required type	Multiple		Description
ref	Reference	false	The reference to one of three following nodes: portalwikis, groupwikis or userwikis.	
owner	String	false	The name of the wiki.	

Properties name	Required type	Multiple		Description
wikiPermissions	String	true	The property consists of the permission information of the wiki. The permission string VIEWPAGE,EDITPAGE,ADMINPAGE,ADMINSPACE:USER:john; VIEWPAGE:GROUP:/platform/users; VIEWPAGE,EDITPAGE,ADMINPAGE,ADMINSPACE:MEMBERSHIP:/platform/administrators.	
isDefaultPermission	Boolean	true	Check whether the default permission is applied to all the wiki tree or not. Its default value is "false".	

WikiHome

The WikiHome node stores the root page of a Wiki. It has the node type "exo:wikihome" that has the super type "wiki:page" inherited from "nt:folder". The super type has the following properties:

Properties name	Required type	Multiple	Description
owner	String	false	The creator of the page.
author	String	false	The last person who modifies the page.
createdDate	Date	false	The date when the page is created.
updatedDate	Date	false	The last date when the page is updated.
syntax	String	false	The Wiki syntax is used to write the page.
title	String	false	The title of the page.
comment	String	false	The comment explains what is modified in the page.

Properties name	Required type	Multiple	Description
url	String	false	The URL to the page.
isOverridePermission	Boolean	false	Check whether the default permission is overridden on the page or not. Its default value is "false".
exo:relation	Reference	true	The property consists of the UUIDs of the related pages.

The nodes that have the type "wiki:page", have a child node named "content" and other child nodes including attachments with the type "wiki:attachment" inherited from "nt:file". The node type "wiki:attachment" has the following properties:

Properties name	Required type	Multiple	Description
title	String	false	The name of the attachment.
fileType	String	false	The type of the attachment.
creator	String	false	The creator of the attachment.

- Three mixin node types: "wiki:removed", "wiki:renamed" and "wiki:watched" may be added to the node type "wiki:page".

The mixin node type "wiki:removed" has the following properties:

Properties name	Required type	Multiple	Description
removedBy	String	false	The person who deleted the page.
removedDate	Date	false	The date when the page is deleted.
parentPath	String	false	The path to the parent page of the deleted page.

The mixin node type "wiki:renamed" has the following properties:

Properties name	Required type	Multiple	Description
oldPageIds	String	true	The old Ids of the renamed page.

The mixin node type "wiki:watched" has the following properties:

Properties name	Required type	Multiple	Description
watcher	String	false	The Id of the person who is watching the page.

Preferences

The "Preferences" node stores the default syntax and page templates of the Wiki. It has the node type "wiki:preferences" and two child nodes "PreferencesSyntax" and "TemplateContainer".

- The "PreferencesSyntax" node has the node type "wiki:preferencessyntax" that has the following properties:

Properties name	Required type	Multiple	Description
defaultSyntax	String	false	The default Wiki syntax of each wiki.
allowMutipleSyntaxes	Boolean	false	Specify whether multiple syntaxes are enabled or not.

- The "TemplateContainer" node stores the page templates. Its child node has the node type "wiki:template" inherited from the node type "wiki:page".

The node type "wiki:template" has the following properties:

Properties name	Required type	Multiple	Description
description	String	false	The description of the template.

LinkRegistry

The "LinkRegistry" node stores the entries to keep track of renaming or moving pages. Each link entry has the node type "wiki:linkentry" that has the following properties:

Properties name	Required type	Multiple	Description
alias	String	false	The Id of a page that is moved or renamed. Its format is "wikitype@wikiowner@page*name".*
newlink	Path	false	A new path to the wiki page that has been moved or renamed.

Trash

The "Trash" node stores deleted pages.

Template Container

The "Template Container" node

Wiki metadata

Wiki metadata are stored in the node wikimetadata that has the node type "wiki:store". These metadata consist of information about help contents, draft contents and the way to find the wikis in the system. The nodes portalwikis, groupwikis and userwikis allow finding the wikis under the type: portal, group and user, basing on the JCR reference features. These nodes have the node types "wiki:portalwikis", "wiki:groupwikis" and "wiki:userwikis" relatively.

Help pages

Help-related information about Wiki syntaxes are stored under the "helppages" node which has the node type "wiki:page". For syntax help, there are two help pages (summary and detailed) for each syntax. The summary page suggests the simple syntax, while the detailed page provides the full syntax-related information. The detailed page is arranged as the child page of the summary page, and both of which have the same node type "wiki:page".

Draft pages

The unsaved content of a new page is stored under the "draftNewPages" node that has the node type "wiki:page".

Developer reference

Extension points

There are some extension points in eXo Knowledge, so that you can control how these components work by implementing or extending default implementations, then reconfigure these new components in the *configuration.xml* file.

ForumEventLifeCycle

Overview

`ForumEventLifeCycle` enables you to listen to the lifecycle of a forum. By implementing ForumEventLifeCycle, you can be notified of new posts and replies, categories and topics. This installation will be injected when the data flow is called to save data.

Configuration plug-in

You can find the configuration file of this component at: *ext/social-integration/src/main/resources/conf/portal/configuration.xml*.

For example, to add a Forum to a space of the Social application and keep new activities of Forum (such as new posts and topics) updated to the activities of space, do as follows:

```
<external-component-plugins>
  <target-component>org.exoplatform.forum.service.ForumService</target-component>
  <component-plugin>
    <name>ForumEventListener</name>
    <set-method>addListenerPlugin</set-method>
    <type>org.exoplatform.ks.ext.impl.ForumSpaceActivityPublisher</type>
  </component-plugin>
</external-component-plugins>
```

Tutorial

To use **ForumEventLifeCycle** class, do the following steps:

1. Create a new class that extends *ForumEventListener*.

For example: class *ABCActivityPublisher*

```
public class ABCActivityPublisher extends ForumEventListener {
.............
}
```

2. Override fuctions in this created class. In each function, you can write anythings to meet your needs.

```
public class ABCActivityPublisher extends ForumEventListener {

  public void saveCategory(Category category){
    ....
  }

  public void saveForum(Forum forum){
    ....
  }

  public void addTopic(Topic topic, String categoryId, String forumId)){
    ....
  }

  public void updateTopic(Topic topic, String categoryId, String forumId){
    ....
  }

  public void addPost(Post post, String categoryId, String forumId, String topicId){
    ....
  }

  public void updatePost(Post post, String categoryId, String forumId, String topicId){
    ....
  }
}
```

- The function *saveCategory* is called when a category is added and/or edited.
- The function *saveForum* is called when a forum is added and/or edited.
- The function *addTopic* is called when a topic is added.
- The function *updateTopic* is called when a topic is updated.

- The function *addPost* is called when a post is added.

- The function *updatePost* is called when a post is updated.

3. Add a new configuration to the *configuration.xml* file with the type that is the class created in the **Step 1**.

```
<external-component-plugins>
  <target-component>org.exoplatform.forum.service.ForumService</target-component>
  <component-plugin>
    <name>ForumEventListener</name>
    <set-method>addListenerPlugin</set-method>
    <type>{package}.{class name}</type>
  <!-- example
  <type>org.exoplatform.ks.ext.impl.ABCActivityPublisher</type>
  -->
  </component-plugin>
</external-component-plugins>
```

AnswerEventLifeCycle

Overview

`AnswerEventLifeCycle` installs event updates for the Answers data that is injected while saving answers, saving questions or posting comments.

Configuration plug-in

You can find the configuration file of this component at:_ext/social-integration/src/main/resources/conf/portal/configuration.xml_.

For example, to add Answers to a space of the Social application and keep new activities of Answers updated to the activities of space, do as follows:

```
<external-component-plugins>
  <target-component>org.exoplatform.faq.service.FAQService</target-component>
  <component-plugin>
    <name>AnswerEventListener</name>
    <set-method>addListenerPlugin</set-method>
    <type>org.exoplatform.ks.ext.impl.AnswersSpaceActivityPublisher</type>
  </component-plugin>
```

```
</external-component-plugins>
```

In which, `AnswersSpaceActivityPublisher` is the class to implement `ForumEventLifeCycle`.

Tutorial

To use the **AnswerEventLifeCycle** class, do the following steps:

1. Create a new class that extends *AnswerEventListener*.

For example: *ABCActivityPublisher*

```
public class ABCActivityPublisher extends AnswerEventListener {
  ....
}
```

2. Override fuctions in this created class. In each function, you can write anything to meet your needs.

```
public class ABCActivityPublisher extends AnswerEventListener {

  public void saveQuestion(Question question, boolean isNew){
    ....
  }

  public void saveAnswer(String questionId, Answer answer, boolean isNew){
    ....
  }

  public void saveAnswer(String questionId, Answer[] answers, boolean isNew){
    ....
  }

  public void saveComment(String questionId, Comment comment, boolean isNew){
    ....
  }
}
```

- The function *saveQuestion* is called when a question is added and/or edited.
- The function *saveAnswer* is called when an answer is added and/or edited.

- The function *saveAnswer* is called when answers are added and/or edited.
- The function *saveComment* is called when a comment is added and/or edited.

3. Add a new configuration to the *configuration.xml* file with the type that is the class created in the **Step 1**.

```
<external-component-plugins>
  <target-component>org.exoplatform.faq.service.FAQService</target-component>
  <component-plugin>
    <name>AnswerEventListener</name>
    <set-method>addListenerPlugin</set-method>
    <type>{package}.{class name}</type>
  <!-- example
  <type>org.exoplatform.ks.ext.impl.ABCActivityPublisher</type>
  -->
  </component-plugin>
</external-component-plugins>
```

BBCodeRenderer

`BBCodeRenderer` is used in the core of eXo Knowledge to render BBCodes. In which, the data input is text, containing BBCode tags. The data output will be BBCode tags which have been encrypted into HTML tags.

You can find the configuration file of this component at: *extension/webapp/src/main/webapp/WEB-INF/ks-extension/ks/forum/bbcodes-configuration.xml*.

For example, to register BBCodeRenderer, do as follows:

```
<external-component-plugins>
                <target-component>org.exoplatform.ks.rendering.MarkupRenderingService</target-component>
  <component-plugin>
    <name>BBCodeRenderer</name>
    <set-method>registerRenderer</set-method>
    <type>org.exoplatform.ks.rendering.spi.RendererPlugin</type>
    <description>BBCode renderer</description>
    <init-params>
      <object-param>
        <name>renderer</name>
        <description>Extended BBCodeRenderer</description>
```

```
      <object type="org.exoplatform.ks.bbcode.core.BBCodeRenderer">
        <field name="bbCodeProvider">
          <object type="org.exoplatform.ks.bbcode.core.ExtendedBBCodeProvider"/>
        </field>
      </object>
    </object-param>
  </init-params>
 </component-plugin>
</external-component-plugins>
```

In which, `ExtendedBBCodeProvider` is the class to implement `BBCodeProvider`.

Internal API

Applications of eXo Knowledge, such as Forum, Answers and Polls, use REST services to communicate with the server. This section describes REST Services of internal APIs.

Forum application

The Forum application of eXo Knowledge uses the ForumWebservice to provide all APIs to work with Forum, such as filtering IPs, viewing RSS, and more.

Service name	Service URL	Description	Location
ForumWebservice	$portalName/ $restcontextname/ ks/ forum/	Call extended services of the Forum application.	* Maven groupId: org.exoplatform.ks * ArtifactId: exo.ks.web.webservice

- **APIs Usage**

Use the following APIs to build all the functions of the Forum application:

Name	Description	Service URL endpoint	Parameters	Expected values
getMessage	Get all new posts with the number based on the *maxcount* value.	$portalName/ $restcontextname/ ks/forum/ getmessage/	String maxcount	integer
filterIps	Ban IPs with no access to Forum.	$portalName/ $restcontextname/ ks/forum/filter/	String str	String in the IP format.

Name	Description	Service URL endpoint	Parameters	Expected values
filterIpBanForum	Ban IPs with no access to a specific Forum.	$portalName/ $restcontextname/ ks/forum/ filterIpBanforum/	String forumId String str	string in the IP format, string in the Forum format
filterTagNameForum	Do the quick search for tags that returns the list of tags.	$portalName/ $restcontextname/ ks/forum/ filterTagNameForum/	String str String userAndTopicId	Tag name id of user and topic
viewrss	Process the request by users when they want to view RSS.	$portalName/ $restcontextname/ ks/forum/rss/	String resourceid	category id forum id topic id
userrss	Process the request by users when they want to view their own collected RSS.	$portalName/ $restcontextname/ ks/forum/rss/ user/	String resourceid	username id

Answers application

The Answers application of eXo Knowledge uses the FAQWebservice to view RSS.

Service name	Service URL	Description	Location
FAQWebservice	$portalName/ $restcontextname/ ks/ faq	Call the extended services of the Answers application.	* Maven groupId: org.exoplatform.ks * ArtifactId: exo.ks.eXoApplication.faq.service

- **APIs Usage**

Use the following APIs to build all the functions of Answers application:

Name	Description	Service URL endpoint	Parameters	Expected values
viewrss	View RSS	$portalName/ $restcontextname/ ks/faq/rss/	String resourceid	category id, question id

Polls application

The Polls application of eXo Knowledge uses the PollWebservice to create and interact with the Polls gadget.

Service name	Service URL	Description	Location
PollWebservice	$portalName/ $restcontextname/ private/ks/poll	Call extended services of the Polls application.	* Maven groupId: org.exoplatform.ks * ArtifactId: exo.ks.eXoApplication.poll.service

- **APIs Usage**

Use the following APIs to build all functions of the Answers application:

Name	Description	Service URL endpoint	Parameters	Expected values
viewPoll	Return data of the Polls system	$portalName/ $restcontextname/ private/ks/poll/ viewpoll/	String pollId	id of poll
votePoll	Update data for the Polls with the returned id	$portalName/ $restcontextname/ private/ks/poll / votepoll/	String pollId String indexVote	id of poll integer

FAQ Template Configuration

Configuration plug-in

Configuration plug-in is used to automatically set up a default template for the FAQ portlet. When the FAQ service starts, it will get values which are returned from the *TemplatePlugin* component to initialize the template for the FAQ portlet.

The template configuration plug-in is configured in the *templates-configuration.xml* file.

In details:

At runtime of the FAQ Service, *FAQService* component is called, then *templates-configuration.xml* file is executed. The component-plugin named *addTemplatePlugin* will be referred to

*org.exoplatform.faq.service.TemplatePlugin- to execute some objects and create default data for the **Forum** application.*

```
<external-component-plugins>
  <target-component>org.exoplatform.faq.service.FAQService</target-component>
  <component-plugin>
    <name>faq.default.template</name>
    <set-method>addTemplatePlugin</set-method>
    <type>org.exoplatform.faq.service.TemplatePlugin</type>
    <init-params>
      <value-param>
        <name>viewerTemplate</name>
        <value>war:/ks-extension/ks/faq/templates/FAQViewerPortlet.gtmpl</value>
      </value-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

The properties of template configuration plug-in are defined in the *init-params* tag as follows:

```
<init-params>
  <value-param>
    <name>viewerTemplate</name>
    <value>war:/ks-extension/ks/faq/templates/FAQViewerPortlet.gtmpl</value>
  </value-param>
</init-params>
```

Name	Description	Value
viewerTemplate	Path of file template.	war:/ks-extension/ks/faq/templates/FAQViewerPortlet.gtmpl

How to change look and feel

You can change the template FAQ viewer in one of the following two ways:

- By using Plug-in.
- By using the **edit mode** to edit template.

Plug-in

1. Create a file named **FAQViewerPortlet.gtmpl** . The content of the file is the template of the FAQ viewer.

2. Copy this file and paste into *ks-extension/WEB-INF/ks-extension/ks/faq/templates/* that is in the **webapps** folder of the server (tomcat, jboss).

When the server runs, *FAQViewerPortlet.gtmpl* will initialize the template of the FAQ viewer.

Edit Mode

1. Run the server and open the FAQ Portlet.

2. Go to **edit mode** and open the **Edit Template** tab.

3. Edit the content of **text-area-input** and click **Save**.

API provided by the UIComponent (UIViewer.java)

- UIViewer is the child of the component *UIFAQPortlet*. It shows the main content of FAQ portlet.

- List of APIs:

Function name	Param	Return	Description
getCategoryInfo	Empty	CategoryInfo object	Get the object **CategoryInfo**
arrangeList	`(List< String > list)`: List of path	A new list is arranged	Arrange a list of path.
render	`(String)`: The content of answers or comments	A new string is converted by function render	Render the content of answers or comments.

- The **CategoryInfo** class:

```
...
private String id;
private String path;
private String name;
private List<String> pathName;
private List<QuestionInfo> questionInfos = new ArrayList<QuestionInfo>();
private List<SubCategoryInfo> subCateInfos = new ArrayList<SubCategoryInfo>();
...
```

Param	Type	Description
id	`String`	The jcr node name of the category node.
path	`String`	The jcr node path of the category node.
name	`String`	The name of the category.
pathName	`List<String>`	The path to the category includes a list of category names.
questionInfos	`List<QuestionInfo>`	The list of **QuestionInfo** object.
subCateInfos	`List<SubCategoryInfo>`	The list of **SubCategoryInfo** object.

- The **QuestionInfo** class:

```
...
private String id;
private String question;
private String detail;
private List<String> answers = new ArrayList<String>();
...
```

Param	Type	Description
id	`String`	The jcr node name of the question node.
question	`String`	The content of the question.
details	`String`	Details of the question.
answers	`List<String>`	The list of answers for the question.

- The **SubCategoryInfo** class:

The params of this class are the same as those of the CategoryInfo class. See *here* for more information.

eXo Social Reference Guide

Applications

List of Portlets in Social

All the portlets are in the *social-portlet.war* file.

Portlet name	Description
Members Portlet	Enable users to search for Space members or list Space members in the alphabetical order.
My Spaces Portlet	Display the spaces that user is member or manager.
Space Activity Stream Portlet	Share Spaces activities.
Invitations Portlet	List all people that invite users.
Requests Portlet	List all invitations requested by users.
Invitation Spaces Portlet	Display the spaces that user is invited.
Pending Spaces Portlet	Display the **Space requests to join** page.
Public Spaces Portlet	Display the **Public Spaces** page.
User Activity Stream Portlet	Update and share the user's activities and/or status.
People Portlet	Display the **People** page.
Connections Portlet	Display the **Connections** page.
User Profile Portlet	Display the **User profile** page.

List of Gadgets in Social

All eXo Social gadgets are in *opensocial.war*.

Activity Stream

- **Description**: Manage activities of users: update status, like/unlike activities, comment activities, delete activities and delete comments.
- **Used REST services**: ActivitiesRestServices.

Social RSS Reader

- **Description**: Fetch, parse and display RSS from a specific URL.
- **Description of user preferences**: There are 2 preference fields: URL input box (default value is *http://blog.exoplatform.org/feed/)* and Number of RSS per page selector (default value is 10).

- **Used REST services**: N/A.

My Connections

- **Description**: Get and display information of the current viewer and his connections.
- **Description of user preferences**: The number of connections displayed per page. It is set to '5' by default.
- **Used REST services**: N/A.

My Spaces

- **Description**: Display all spaces that a user has the "member" role.
- **Used REST services**: SpacesRestService.

Configuration

Components

ActivityManager

Configuration name	Data type	Possible value	Description
allowedTags	`String list`	`b, i, a, span, em, strong, p, ol, ul,li, br, img`	HTML tags

SpaceService

Configuration name	Data type	Possible value	Description
space.homeNodeApp	`String`	`SpaceActivityStreamPortlet`	The home application for a space.
space.apps	`String list`	`DashboardPortlet:true, SpaceSettingPortlet:false, MembersPortlet:true`	The applications that are used for initializing the application when the space is created.

Note

Deprecated: Use external-component-plugins instead: org.exoplatform.social.core.space.SpaceApplicationConfigPlugin.

IdentityManager

Configuration name	Data type	Possible value	Description
providers	`String`	`org.exoplatform.social.core.identity.provider.SpaceIden`	The identity providers.

ProfileConfig

Configuration name	Data type	Possible value	Description
nodetype.emails	`String`	`exo:profileKeyValue`	
nodetype.phones	`String`	`exo:profileKeyValue`	
nodetype.ims	`String`	`exo:profileKeyValue`	
nodetype.urls	`String`	`exo:profileKeyValue`	
nodetype.address	`String`	`exo:profileAddress`	
nodetype.experiences	`String`	`exo:profileExperience`	
nodetype.education	`String`	`exo:profileEducation`	

Configuration name	Data type	Possible value	Description
forceMultiValue	`String`	xxxxxxxxxxxx	

ServiceProviderStore

Configuration name	Data type	Possible value	Description
sample-provider	properties-params		Sample service provider.

External component plugins

MentionsProcessor

Configuration name	Data type	Possible value	Description
priority	`String`	2	The priority of this processor (the lower is executed first).

PortletPreferenceRequiredPlugin

Configuration name	Data type	Possible value	Description
portletsPrefsRequired	`String list`	`SpaceActivityStreamPortlet`, `SpaceSettingPortlet`, `MembersPortlet`	The portlet name, which requires space URL preference.

SpaceApplicationConfigPlugin

Configuration name	Data type	Description
spaceHomeApplication	`SpaceApplicationConfigPlugin`	The space application for the space home node.
spaceApplicationList	`SpaceApplicationConfigPlugin`	Space application list configuration.

AddNodeTypePlugin

Configuration name	Data type	Possible value	Description
autoCreatedInNewRepository	`String`	`jar:/conf/portal/core-nodetypes.xml`	Node types configuration file.

RelationshipManager

The Service is used to manipulate user relationships.

SpaceIdentityProvider

The provider is to give identity space instances.

SpaceApplicationHandler

The service is to handle all events related to creating and managing all the application spaces.

ExoPeopleService

The service is to manipulate all data related to people in the Portal.

Space Service

Description

The service for spaces management includes creating spaces, and installing applications.

Components configuration

Component name	Description
SpaceServiceImpl	Implementation class of Space Service.

```
<component>
  <key>org.exoplatform.social.core.space.spi.SpaceService</key>
  <type>org.exoplatform.social.core.space.impl.SpaceServiceImpl</type>
  <!--Deprecated, Use external-component-plugins instead
  <init-params>
    <values-param>
      <name>space.homeNodeApp</name>
      <value>SpaceActivityStreamPortlet</value>
    </values-param>
    <values-param>
      <name>space.apps</name>
      <value>DashboardPortlet:true</value>
      <value>SpaceSettingPortlet:false</value>
      <value>MembersPortlet:true</value>
    </values-param>
  </init-params>
  -->
</component>
```

Configuration name	Data type	Possible value	Possible value	Description
SpaceActivityStreamPortlet	`String`	N/A	`SpaceActivityStreamPortlet`	The name of portlet displaying activities of spaces.
space.apps	`String list`	`Porlets' name: true/false`	`DashboardPortlet:true; SpaceSettingPortlet:true, MembersPortlet:true`	list of configurations for portlets.

External plug-in configuration

PortletPreferenceRequiredPlugin

```
<external-component-plugins>
  <target-component>org.exoplatform.social.core.space.spi.SpaceService</target-component>
  <component-plugin>
    <name>portlets.prefs.required</name>
    <set-method>setPortletsPrefsRequired</set-method>
    <type>org.exoplatform.social.core.application.PortletPreferenceRequiredPlugin</type>
    <init-params>
      <values-param>
        <name>portletsPrefsRequired</name>
        <value>SpaceActivityStreamPortlet</value>
        <value>SpaceSettingPortlet</value>
        <value>MembersPortlet</value>
      </values-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

In which:

Name	Set-method	Type	Description
PortletPreferenceRequiredPlugin	setPortletsPrefsRequired	org.exoplatform.social.core.application.PortletPreferenceRequiredPlugin	Configuration the list of portlet names which will have portlet preference of space context.

- Init-params:

Name	Possible value	Possible value	Description
portletsPrefsRequired	`Portlet names`	`SpaceActivityStreamPortlet`, `SpaceSettingPortlet`, `MembersPortlet`	List of portlets which need to be saved and get the space context name.

SpaceApplicationConfigPlugin

Note

Since 1.2.0-GA

```
<external-component-plugins>
  <target-component>org.exoplatform.social.core.space.spi.SpaceService</target-component>
  <!-- Default applications to be installed when creating a new space -->
  <component-plugin>
    <name>Space Application Configuration</name>
    <set-method>setSpaceApplicationConfigPlugin</set-method>
    <type>org.exoplatform.social.core.space.SpaceApplicationConfigPlugin</type>
    <init-params>
      <object-param>
        <name>spaceHomeApplication</name>
        <description>Space Home Application</description>
                                                                                <object
 type="org.exoplatform.social.core.space.SpaceApplicationConfigPlugin$SpaceApplication">
          <field name="portletApp">
            <string>social-portlet</string>
          </field>
          <field name="portletName">
            <string>SpaceActivityStreamPortlet</string>
          </field>
          <field name="appTitle">
            <string>Home</string>
          </field>
          <!--<field name="icon"><string>SpaceHomeIcon</string></field> -->
        </object>
      </object-param>
      <object-param>
        <name>spaceApplicationListConfig</name>
        <description>space application list configuration</description>
        <object type="org.exoplatform.social.core.space.SpaceApplicationConfigPlugin">
          <field name="spaceApplicationList">
            <collection type="java.util.ArrayList">
```

```
        <value>
                                                                                <object
type="org.exoplatform.social.core.space.SpaceApplicationConfigPlugin$SpaceApplication">
            <field name="portletApp">
              <string>dashboard</string>
            </field>
            <field name="portletName">
              <string>DashboardPortlet</string>
            </field>
            <field name="appTitle">
              <string>Dashboard</string>
            </field>
            <field name="removable">
              <boolean>true</boolean>
            </field>
            <field name="order">
              <int>1</int>
            </field>
            <field name="uri">
              <string>dashboard</string>
            </field>
            <!--<field name="icon"><string>SpaceDashboardIcon</string></field> -->
          </object>
        </value>
        <value>
                                                                                <object
type="org.exoplatform.social.core.space.SpaceApplicationConfigPlugin$SpaceApplication">
            <field name="portletApp">
              <string>social-portlet</string>
            </field>
            <field name="portletName">
              <string>SpaceSettingPortlet</string>
            </field>
            <field name="appTitle">
              <string>Space Settings</string>
            </field>
            <field name="removable">
              <boolean>false</boolean>
            </field>
            <field name="order">
              <int>2</int>
            </field>
            <field name="uri">
              <string>settings</string>
```

```
            </field>
            <!--<field name="icon"><string>SpaceSettingsIcon</string></field> -->
          </object>
        </value>
        <value>
                                                                    <object
type="org.exoplatform.social.core.space.SpaceApplicationConfigPlugin$SpaceApplication">
            <field name="portletApp">
              <string>social-portlet</string>
            </field>
            <field name="portletName">
              <string>MembersPortlet</string>
            </field>
            <field name="appTitle">
              <string>Members</string>
            </field>
            <field name="removable">
              <boolean>true</boolean>
            </field>
            <field name="order">
              <int>3</int>
            </field>
            <field name="uri">
              <string>members</string>
            </field>
            <!--<field name="icon"><string>SpaceMembersIcon</string></field> -->
          </object>
        </value>
      </collection>
    </field>
  </object>
 </object-param>
</init-params>
</component-plugin>
</external-component-plugins>
```

In which:

Name	Set-method	Type	Description
Space Application Configuration	setSpaceApplicationConfigPlugin	org.exoplatform.social.core.application.PortletPreferenceRequiredPlugin	Configure the list of space applications to be installed when creating a new space.

Activity Manager

Description

The service is to manipulate space and user activities.

Component plug-in configuration

```
<component-plugin>
  <name>OSHtmlSanitizer</name>
  <set-method>addProcessorPlugin</set-method>
  <type>org.exoplatform.social.core.processor.OSHtmlSanitizerProcessor</type>
  <init-params>
    <values-param>
      <name>allowedTags</name>
      <value>b</value>
      <value>i</value>
      <value>span</value>
      <value>a</value>
      <value>em</value>
      <value>strong</value>
      <value>p</value>
      <value>ol</value>
      <value>ul</value>
      <value>li</value>
      <value>br</value>
      <value>img</value>
    </values-param>
  </init-params>
</component-plugin>
```

In which,

Name	Set-method	Type	Description
OSHtmlSanitizerProcessor	addProcessorPlugin	org.exoplatform.social.core.processor.OSHtmlSanitizerProcessor	The plugin that renders valid HTML tags appearing in the Activity body (content).

- Init-params:

Name	Possible value	Possible value	Description
allowedTags	`html tags`	b, i, a, span, em, strong, p, ol, ul,li, br, img	Process and render HTML tags in the activity content (body).

External plug-in configuration

```
<component-plugin>
  <name>MentionsProcessor</name>
  <set-method>addProcessorPlugin</set-method>
  <type>org.exoplatform.social.core.processor.MentionsProcessor</type>
  <init-params>
    <value-param>
      <name>priority</name>
      <description>priority of this processor (lower number will be executed first)</description>
      <value>2</value>
    </value-param>
  </init-params>
</component-plugin>
```

In which:

Name	Set-method	Type	Description
MentionsProcessor	addProcessorPlugin	org.exoplatform.social.core.processor.MentionsProcessor	A processor that will substitute @username expressions by a link on the user profile.

- Init-params:

Name	Possible value	Possible value	Description
priority	`priority number`	2	The priority of this processor (The lower level will be executed first).

Identity Manager

Description

The service is to manipulate the identity operations, such as creating, getting, deleting or finding a profile.

Component plug-in configuration

```
<component>
  <key>org.exoplatform.social.core.manager.IdentityManager</key>
  <type>org.exoplatform.social.core.manager.IdentityManager</type>
  <component-plugins>
    <component-plugin>
      <name>SpaceIdentityProvider plugin</name>
      <set-method>registerIdentityProviders</set-method>
      <type>org.exoplatform.social.core.identity.IdentityProviderPlugin</type>
      <init-params>
        <values-param>
          <name>providers</name>
          <description>Identity Providers</description>
          <value>org.exoplatform.social.core.identity.provider.SpaceIdentityProvider</value>
        </values-param>
      </init-params>
    </component-plugin>
  </component-plugins>
</component>
```

In which:

Name	Set-method	Type	Description
SpaceIdentityProvider plugin	registerIdentityProviders	org.exoplatform.social.core.identity.IdentityProviderPlugin	The plugin provides identity for a space.

- Init-params:

Name	Possible value	Possible value	Description
providers	`Every other identity providers`	`org.exoplatform.social.core.identity.provider.SpaceIden`	

Name	Possible value	Possible value	Description
			Identity Provider instances for managing identities.

OpenSocial Rest Context Configuration

Description

The service is used to configure the portal container name when there is an OpenSocial REST API request. By configuring this service, you can make sure that the right portal container is reached. The default portal container is *portal*.

Component plug-in configuration

This should be used when there is a portal container different than the default *portal* one.

```
<external-component-plugins>
  <target-component>org.exoplatform.social.opensocial.auth.RestPortalContainerNameConfig</
target-component>
 <component-plugin>
   <name>set portal container name used for REST service</name>
   <set-method>setRestContainerName</set-method>
   <type>org.exoplatform.social.opensocial.auth.RestPortalContainerNameConfig</type>
   <init-params>
     <value-param>
       <name>rest-container-name</name>
       <value>socialdemo</value>
     </value-param>
   </init-params>
 </component-plugin>
</external-component-plugins>
```

In which:

Name	Set-method	Type	Description
set portal container name used for REST service	setRestContainerName	org.exoplatform.social.opensocial.auth.RestPortalContainerNameC	This plugin is used to set the rest portal container name for the OpenSocial REST API request.

- Init-params:

Name	Possible value	Possible value	Description
rest-container-name	`any valid portal container name`	`N/A`	Name of the portal container.

Spaces Template configuration

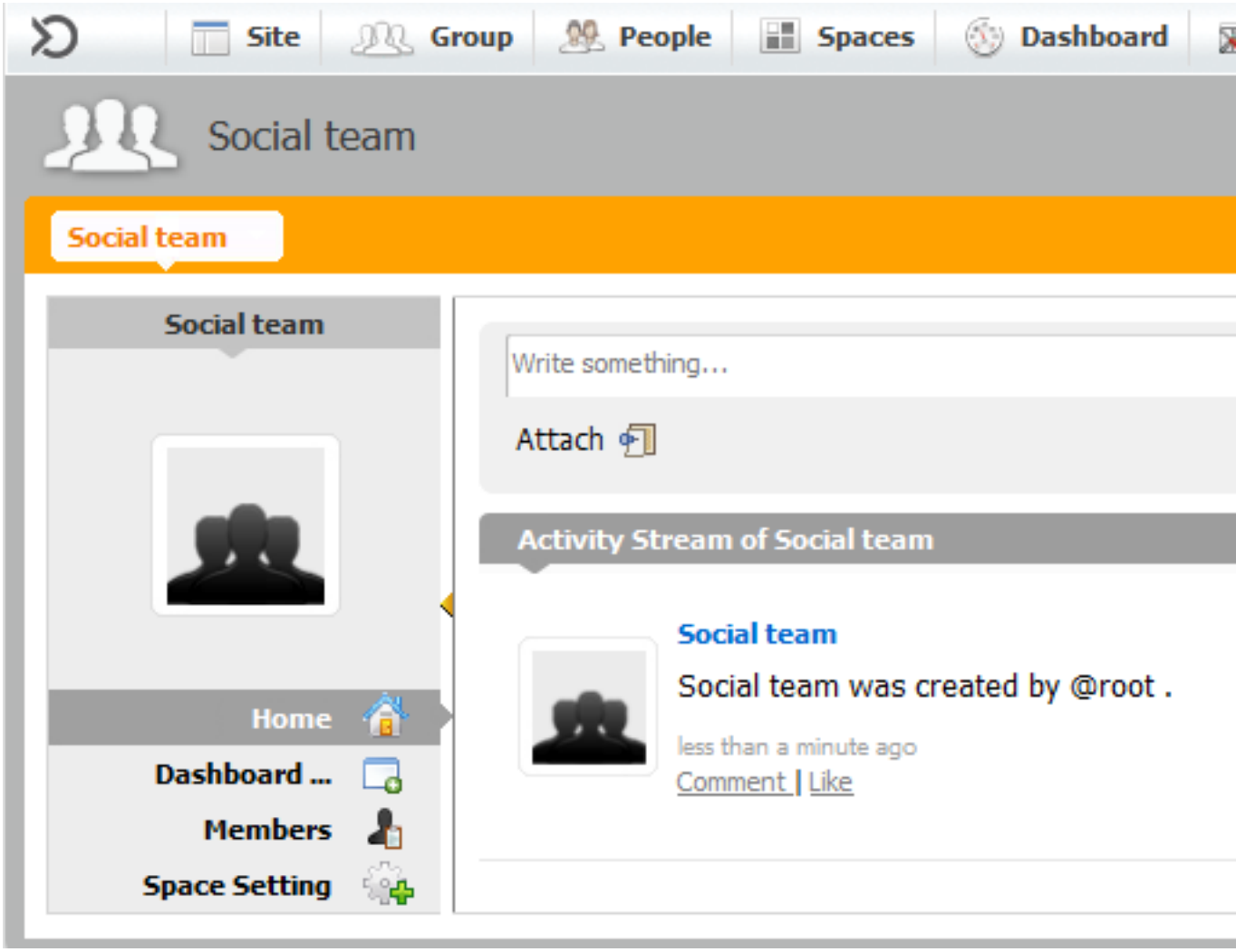
In eXo Social, you may have two space types (classic and webos spaces).

For the classic space, you can pre-configure the template. You can configure the layout to select where to display the applications (for example, the application's menu on the left, the selected application is displayed on the right, and more).

Here is an example of the configuration file that displays the menu on the left. The application will be inserted in the container with the ID **Application**:

```
<page>
  <owner-type/>
  <owner-id/>
  <name/>
            <container id="SpacePage" template="system:/groovy/portal/webui/container/
UITableColumnContainer.gtmpl">
    <container id="Menu" template="system:/groovy/portal/webui/container/UIContainer.gtmpl">
      <portlet-application>
        <portlet>
          <application-ref>space</application-ref>
          <portlet-ref>SpaceMenuPortlet</portlet-ref>
        </portlet>
        <access-permissions>*:/platform/users</access-permissions>
        <show-info-bar>false</show-info-bar>
      </portlet-application>
    </container>
                  <container id="Application" template="system:/groovy/portal/webui/container/
UIContainer.gtmpl">
    </container>
  </container>
</page>
```

In this example, the outer container contains two inner containers: one container has id as **Menu** for your Menu and another has id as **Application** containing your applications.



If you want to put the menu on the right and the application on the left, you can swap the declared position of these two containers:

```
<page>
  <owner-type/>
  <owner-id/>
  <name/>
            <container       id="SpacePage"       template="system:/groovy/portal/webui/container/
UITableColumnContainer.gtmpl">
                  <container     id="Application"     template="system:/groovy/portal/webui/container/
UIContainer.gtmpl">
    <container id="Menu" template="system:/groovy/portal/webui/container/UIContainer.gtmpl">
      <portlet-application>
        <portlet>
          <application-ref>space</application-ref>
          <portlet-ref>SpaceMenuPortlet</portlet-ref>
```

```
      </portlet>
      <access-permissions>*:/platform/users</access-permissions>
      <show-info-bar>false</show-info-bar>
    </portlet-application>
  </container>
  </container>
 </container>
</page>
```

In eXo Social standalone, this configuration file is at:

- *$EXO_TOMCAT/webapps/socialdemo/WEB-INF/conf/portal/template/pages/space/page.xml*

In eXo Platform, this configuration file is at:

- *$EXO_TOMCAT/webapps/{portal-name}/WEB-INF/conf/portal/template/pages/space/page.xml*

Configure the oauth 2 legged scenario

This section is about configuring the oAuth 2 legs scenario in OpenSocial. (Reference: *OpenSocial.org* [http://docs.opensocial.org/display/OS/Home])

For more information, visit *2-legged OAuth for the OpenSocial REST API.* [http://sites.google.com/site/oauthgoog/2leggedoauth/2opensocialrestapi]

Generate the certificates

To generate the key:

```
$ openssl req -newkey rsa:1024 -days 365 -nodes -x509 -keyout testkey.pem
    -out testkey.pem -subj '/CN=mytestkey'
$ openssl pkcs8 -in testkey.pem -out oauthkey.pem -topk8 -nocrypt -outform PEM
```

Configure the property file

Edit *container.js* and change the following parameter to point to your private key and key name.

```
"gadgets.signingKeyFile" : "oauth.pem",
"gadgets.signingKeyName" : "oauthKey",
```

Developers References

UI Extensions

About Activity Plugin

The activity plugin feature was introduced to allow using the activity composer extension and the custom UI component for displaying activity by its type.

How to create activity plugin

You should have an idea about the UI Extension Framework. If you have already worked with the UI Extension Framework, it is really easy to create an activity plugin. If not, you have chance to work with it now. You should have a look at *UI Extension Framework* [http://wiki.exoplatform.org/xwiki/bin/view/ECM/UI+Extension+Framework].

Create a custom UI component for displaying the activity based on its type

Note

The Project Code is available *here.* [http://wiki.exoplatform.org/xwiki/bin/download/Social/Developer%20Documentation%20Activity%20Plugin/exo.social.samples.activity-plugin.zip]

When an activity is displayed, UIActivityFactory will look for its registered custom actvity display by activity's type. If not found, UIDefaultActivity will be called for displaying that activity.

For example, in eXo Social, there is an activity of type "exosocial:spaces" created by SpaceActivityPublisher, but now, you want to display it without own UI component instead of the default one. To do that, follow these steps.

1. Create a sample project:

```
mvn archetype:generate
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'archetype'.
[INFO] artifact org.apache.maven.plugins:maven-archetype-plugin: checking
 for updates from central
[INFO] snapshot
 org.apache.maven.plugins:maven-archetype-plugin:2.0-alpha-6-SNAPSHOT:
 checking for updates from central
[INFO] snapshot
 org.apache.maven.archetype:maven-archetype:2.0-alpha-6-SNAPSHOT: checking
 for updates from central
```

```
[INFO]
 ------------------------------------------------------------------------
[INFO] Building Maven Default Project
[INFO]    task-segment: [archetype:generate] (aggregator-style)
[INFO]
 ------------------------------------------------------------------------
[INFO] Preparing archetype:generate
[INFO] No goals needed for project - skipping
[INFO] snapshot
 org.apache.maven.archetype:archetype-common:2.0-alpha-6-SNAPSHOT: checking
 for updates from central
[INFO] snapshot
 org.apache.maven.archetype:archetype-common:2.0-alpha-6-SNAPSHOT: checking
 for updates from apache.snapshots
[INFO] [archetype:generate {execution: default-cli}]
[INFO] Generating project in Interactive mode
[INFO] No archetype defined. Using maven-archetype-quickstart
 (org.apache.maven.archetypes:maven-archetype-quickstart:1.0)
Choose archetype:
1: remote -> docbkx-quickstart-archetype (null)
2: remote -> gquery-archetype (null)
3: remote -> gquery-plugin-archetype (null)
//....

285: remote -> wicket-scala-archetype (Basic setup for a project that
 combines Scala and Wicket,
  depending on the Wicket-Scala project. Includes an example Specs
  test.)
286: remote -> circumflex-archetype (null)
Choose a number: 76: 76
Choose version:
1: 1.0
2: 1.0-alpha-1
3: 1.0-alpha-2
4: 1.0-alpha-3
5: 1.0-alpha-4
6: 1.1
Choose a number: : 1
Define value for property 'groupId': : org.exoplatform.social.samples
Define value for property 'artifactId': : exo.social.samples.activity-plugin

Define value for property 'version': 1.0-SNAPSHOT: 1.0.0-SNAPSHOT
Define value for property 'package': org.exoplatform.social.samples:
 org.exoplatform.social.samples.activityPlugin
Confirm properties configuration:
groupId: org.exoplatform.social.samples
artifactId: exo.social.samples.activity-plugin
version: 1.0.0-SNAPSHOT
package: org.exoplatform.social.samples.activityPlugin
```

```
Y: y
```

2. Edit the *pom.xml* file as below.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.exoplatform.social</groupId>
    <artifactId>social-project</artifactId>
    <version>1.1.0-GA</version>
  </parent>
  <groupId>org.exoplatform.social.samples</groupId>
  <artifactId>exo.social.samples.activity-plugin</artifactId>
  <packaging>jar</packaging>
  <version>1.1.0-GA</version>
  <name>exo.social.samples.activity-plugin</name>
  <build>
    <sourceDirectory>src/main/java</sourceDirectory>
    <outputDirectory>target/classes</outputDirectory>
    <resources>
      <resource>
        <directory>src/main/resources</directory>
        <includes>
          <include>**/*.gtmpl</include>
        </includes>
      </resource>
      <resource>
        <directory>src/main/java</directory>
        <includes>
          <include>**/*.xml</include>
        </includes>
      </resource>
    </resources>
  </build>
  <dependencies>
    <dependency>
      <groupId>org.exoplatform.portal</groupId>
      <artifactId>exo.portal.webui.core</artifactId>
      <scope>provided</scope>
    </dependency>
    <dependency>
```

```
    <groupId>org.exoplatform.portal</groupId>
    <artifactId>exo.portal.webui.portal</artifactId>
    <scope>provided</scope>
   </dependency>
   <dependency>
    <groupId>org.exoplatform.social</groupId>
    <artifactId>exo.social.component.core</artifactId>
    <scope>provided</scope>
   </dependency>
   <dependency>
    <groupId>org.exoplatform.social</groupId>
    <artifactId>exo.social.component.webui</artifactId>
    <scope>provided</scope>
   </dependency>
   <dependency>
    <groupId>org.exoplatform.social</groupId>
    <artifactId>exo.social.component.service</artifactId>
    <scope>provided</scope>
   </dependency>
  </dependencies>
</project>
```

To use the custom UI component for displaying its activity, you need a class that must be a subclass of *BaseUIActivity*.

3. Call *UISpaceSimpleActivity*

```
package org.exoplatform.social.samples.activityplugin;

import org.exoplatform.social.webui.activity.BaseUIActivity;
import org.exoplatform.webui.config.annotation.ComponentConfig;
import org.exoplatform.webui.core.lifecycle.UIFormLifecycle;

@ComponentConfig(
  lifecycle = UIFormLifecycle.class,
  template = "classpath:groovy/social/plugin/space/UISpaceSimpleActivity.gtmpl"
)
public class UISpaceSimpleActivity extends BaseUIActivity {

}
```

The *UISpaceSimpleActivity.gtmpl* template should be created under *main/resources/groovy/social/plugin/space*:

```
<div>This is a space activity UI component displayed for type "exosocial:spaces"</div>
```

An activity builder as *explained later* is also needed.

```
package org.exoplatform.social.samples.activityplugin;

import org.exoplatform.social.core.activity.model.Activity;
import org.exoplatform.social.webui.activity.BaseUIActivity;
import org.exoplatform.social.webui.activity.BaseUIActivityBuilder;

public class SimpleSpaceUIActivityBuilder extends BaseUIActivityBuilder {

  @Override
  protected void extendUIActivity(BaseUIActivity uiActivity, Activity activity) {
    // TODO Auto-generated method stub

  }

}
```

4. Create the *configuration.xml* file under *conf/portal*:

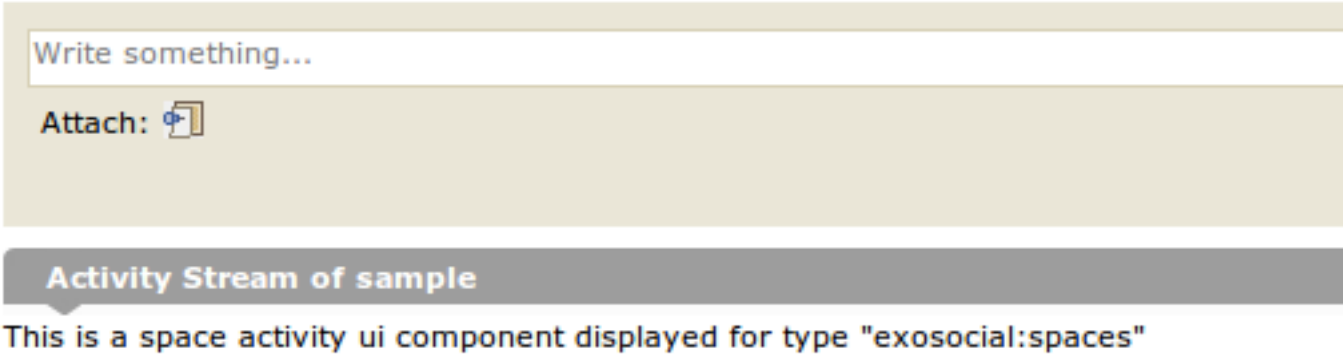
```
<configuration xmlns="http://www.exoplaform.org/xml/ns/kernel_1_1.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.exoplaform.org/xml/ns/kernel_1_1.xsd http://www.exoplaform.org/xml/ns/kernel_1_1.xsd">
  <external-component-plugins>
    <target-component>org.exoplatform.webui.ext.UIExtensionManager</target-component>
    <component-plugin>
      <name>add.action</name>
      <set-method>registerUIExtensionPlugin</set-method>
      <type>org.exoplatform.webui.ext.UIExtensionPlugin</type>
      <init-params>
        <object-param>
          <name>Simple Space Activity</name>
          <object type="org.exoplatform.social.webui.activity.UIActivityExtension">
```

```
          <field name="type">
            <string>org.exoplatform.social.webui.activity.BaseUIActivity</string>
          </field>
          <field name="name">
            <string>exosocial:spaces</string>
          </field>
          <field name="component">
            <string>org.exoplatform.social.samples.activityplugin.UISpaceSimpleActivity</string>
          </field>
          <field name="activityBuiderClass">
              <string>org.exoplatform.social.samples.activityplugin.SimpleSpaceUIActivityBuilder</
string>
          </field>
        </object>
      </object-param>
    </init-params>
  </component-plugin>
 </external-component-plugins>
</configuration>
```

Note that *exosocial:spaces* must have its value matching with the activity's type that you want to display with your UI component.

Assume that you have already built the Social project with version 1.1. If you do not know how to, have a look at *Building from Sources* [Main.Building from sources] with *Social 1.1.0-CR01* [http://svn.exoplatform.org/projects/social/tags/1.1.0-CR01]. Next, build a sample project and copy the jar file to /tomcat/lib. Then, run Social, create a space and access it, you can see the space's activity of type "exosocial:spaces" is displayed by default in Social:

The custom UI component for displaying activity of type "exosocial:spaces" is like below:



5. Make the custom UI activity display have the look, feel and function like the default one.

When displaying an activity, you should make sure that the look and feel of the custom UI component is consistent and match with other activities and have the full functions of **Like**, **Comments**. To create another UI component to display, call *UISpaceLookAndFeelActivity*:

```
package org.exoplatform.social.samples.activityplugin;

import org.exoplatform.social.webui.activity.BaseUIActivity;
import org.exoplatform.webui.config.annotation.ComponentConfig;
import org.exoplatform.webui.config.annotation.EventConfig;
import org.exoplatform.webui.core.lifecycle.UIFormLifecycle;

@ComponentConfig(
 lifecycle = UIFormLifecycle.class,
 template = "classpath:groovy/social/plugin/space/UISpaceLookAndFeelActivity.gtmpl",
 events = {
   @EventConfig(listeners = BaseUIActivity.ToggleDisplayLikesActionListener.class),
   @EventConfig(listeners = BaseUIActivity.ToggleDisplayCommentFormActionListener.class),
   @EventConfig(listeners = BaseUIActivity.LikeActivityActionListener.class),
   @EventConfig(listeners = BaseUIActivity.SetCommentListStatusActionListener.class),
   @EventConfig(listeners = BaseUIActivity.PostCommentActionListener.class),
       @EventConfig(listeners = BaseUIActivity.DeleteActivityActionListener.class, confirm =
"UIActivity.msg.Are_You_Sure_To_Delete_This_Activity"),
     @EventConfig(listeners = BaseUIActivity.DeleteCommentActionListener.class, confirm =
"UIActivity.msg.Are_You_Sure_To_Delete_This_Comment")
 }
)
public class UISpaceLookAndFeelActivity extends BaseUIActivity {
```

```
}
```

6. Create the *UISpaceLookAndFeelActivity* template. The easiest way is to copy the content of *UIDefaultActivity.gtmpl* to this template file:

```
<%
  import org.exoplatform.portal.webui.util.Util;
  import org.exoplatform.webui.form.UIFormTextAreaInput;

  def pcontext = Util.getPortalRequestContext();
  def jsManager = pcontext.getJavascriptManager();
  def labelActivityHasBeenDeleted = _ctx.appRes("UIActivity.label.Activity_Has_Been_Deleted");
  def activity = uicomponent.getActivity();
  def activityDeletable = uicomponent.isActivityDeletable();
%>

<% if (activity) { //process if not null

  jsManager.importJavascript("eXo.social.Util", "/social-resources/javascript");
  jsManager.importJavascript("eXo.social.PortalHttpRequest", "/social-resources/javascript");
  jsManager.importJavascript("eXo.social.webui.UIForm", "/social-resources/javascript");
  jsManager.importJavascript("eXo.social.webui.UIActivity", "/social-resources/javascript");

  def labelComment = _ctx.appRes("UIActivity.label.Comment");
  def labelLike = _ctx.appRes("UIActivity.label.Like");
  def labelUnlike = _ctx.appRes("UIActivity.label.Unlike");
  def labelSource = _ctx.appRes("UIActivity.label.Source");
  def inputWriteAComment = _ctx.appRes("UIActivity.input.Write_A_Comment");
  def labelShowAllComments = _ctx.appRes("UIActivity.label.Show_All_Comments");
  def labelHideAllComments = _ctx.appRes("UIActivity.label.Hide_All_Comments");
  def labelOnePersonLikeThis = _ctx.appRes("UIActivity.label.One_Person_Like_This");
  def labelPeopleLikeThis = _ctx.appRes("UIActivity.label.People_Like_This");
  def labelYouLikeThis = _ctx.appRes("UIActivity.label.You_Like_This");
                                    def                labelYouAndOnePersonLikeThis                =
 _ctx.appRes("UIActivity.label.You_And_One_Person_Like_This");
  def labelYouAndPeopleLikeThis = _ctx.appRes("UIActivity.label.You_And_People_Like_This");

  def likeActivityAction = uicomponent.event("LikeActivity", "true");
  def unlikeActivityAction = uicomponent.event("LikeActivity", "false");

  def commentList = uicomponent.getComments();
```

```
 def allComments = uicomponent.getAllComments();
 if (allComments) {
   labelShowAllComments = labelShowAllComments.replace("{0}", allComments.size() + "");
   labelHideAllComments = labelHideAllComments.replace("{0}", allComments.size() + "");

 }
 def displayedIdentityLikes = uicomponent.getDisplayedIdentityLikes();
 def identityLikesNum = 0;
 def labelLikes = null;
 def toggleDisplayLikesAction = uicomponent.event("ToggleDisplayLikes");
              def startTag = "<a onclick={{{"}}}$toggleDisplayLikesAction{{{"}}}

href={{{"}}}#ToggleDisplayListPeopleLikes{{{"}}}>";
 def endTag = "</a>";
 if (displayedIdentityLikes != null) {
   identityLikesNum = displayedIdentityLikes.length;
 }
 def commentListStatus = uicomponent.getCommentListStatus();
 def commentFormDisplayed = uicomponent.isCommentFormDisplayed();
 def likesDisplayed = uicomponent.isLikesDisplayed();
 //params for init UIActivity javascript object
 def params = """
   {activityId: '${activity.id}',
    inputWriteAComment: '$inputWriteAComment',
    commentMinCharactersAllowed: ${uicomponent.getCommentMinCharactersAllowed()},
    commentMaxCharactersAllowed: ${uicomponent.getCommentMaxCharactersAllowed()},
    commentFormDisplayed: $commentFormDisplayed,
    commentFormFocused: ${uicomponent.isCommentFormFocused()}
   }
 """
 jsManager.addOnLoadJavascript("initUIActivity${activity.id}");
  //make sures commentFormFocused is set to false to prevent any refresh to focus, only focus
after post a comment
 uicomponent.setCommentFormFocused(false);
     def activityUserName, activityUserProfileUri, activityImageSource, activityContentBody,
activityPostedTime;
 def commentFormBlockClass = "", listPeopleLikeBlockClass = "", listPeopleBGClass = "";
 if (!commentFormDisplayed) {
   commentFormBlockClass = "DisplayNone";
 }

 if (!likesDisplayed) {
   listPeopleLikeBlockClass = "DisplayNone";
 }
```

```
 if (uicomponent.isLiked()) {
   if (identityLikesNum > 1) {
            labelLikes = labelYouAndPeopleLikeThis.replace("{start}", startTag).replace("{end}",
endTag).replace("{0}", identityLikesNum + "");
   } else if (identityLikesNum == 1) {
        labelLikes = labelYouAndOnePersonLikeThis.replace("{start}", startTag).replace("{end}",
endTag);
   } else {
     labelLikes = labelYouLikeThis;
   }
 } else {
   if (identityLikesNum > 1) {
                    labelLikes = labelPeopleLikeThis.replace("{start}", startTag).replace("{end}",
endTag).replace("{0}", identityLikesNum + "");
   } else if (identityLikesNum == 1) {
      labelLikes = labelOnePersonLikeThis.replace("{start}", startTag).replace("{end}", endTag);
   }
 }

 if (!labelLikes) {
  //hides diplayPeopleBG
  listPeopleBGClass = "DisplayNone";
 }

 activityContentTitle = activity.title;
 activityPostedTime = uicomponent.getPostedTimeString(activity.postedTime);
 activityUserName = uicomponent.getUserFullName(activity.userId);
 activityUserProfileUri = uicomponent.getUserProfileUri(activity.userId);

 activityImageSource = uicomponent.getUserAvatarImageSource(activity.userId);
 if (!activityImageSource)  {
                  activityImageSource     =     "/social-resources/skin/ShareImages/SpaceImages/
SpaceLogoDefault_61x61.gif";
 }

%>

<div class="UIActivity">
 <script type="text/javascript">
   function initUIActivity${activity.id}() {
     new eXo.social.webui.UIActivity($params);
   }
 </script>
```

```
<% uiform.begin() %>
<div class="NormalBox clearfix">
  <a class="Avatar" title="$activityUserName" href="$activityUserProfileUri">
        <img title="$activityUserName" src="$activityImageSource" alt="$activityUserName"
height="54px" width="56px">
  </a>
  <div class="ContentBox" id="ContextBox${activity.id}">
    <div class="TitleContent clearfix">
      <div class="Text">
        <a title="$activityUserName" href="$activityUserProfileUri">$activityUserName</a>
      </div>
    <% if (activityDeletable) {%>
        <div onclick="<%= uicomponent.event("DeleteActivity", uicomponent.getId(), ""); %>"
class="CloseContentBoxNormal" id="DeleteActivityButton${activity.id}"><span></span></div>
    <%}%>
    </div>
    <div class="Content">
      $activityContentTitle (from custom UI component)<br>
    </div>
    <div class="LinkCM">
      <span class="DateTime">$activityPostedTime *</span>
     <% def toggleDisplayCommentAction = uicomponent.event('ToggleDisplayCommentForm',
null, false);
       def commentLink = "";
    %>
                <a class="LinkCM $commentLink" onclick="$toggleDisplayCommentAction"
id="CommentLink${activity.id}" href="#comment">
        $labelComment
      </a> |
    <% if (uicomponent.isLiked()) { %>
              <a onclick="$unlikeActivityAction" class="LinkCM" id="UnLikeLink${activity.id}"
href="#unlike">
        $labelUnlike
      </a>
    <% } else { %>
      <a onclick="$likeActivityAction" class="LinkCM" id="LikeLink${activity.id}" href="#like">
        $labelLike
      </a>
    <% }%>
    </div>
  </div>

  <div class="ListPeopleLikeBG $listPeopleBGClass">
```

```
    <div class="ListPeopleLike">
      <div class="ListPeopleContent">
      <% if (!labelLikes) labelLikes = ""; %>
        <div class="Title">$labelLikes</div>
        <div class="$listPeopleLikeBlockClass">
        <%
        //def personLikeFullName, personLikeProfileUri, personLikeAvatarImageSource;

        displayedIdentityLikes.each({
          personLikeFullName = uicomponent.getUserFullName(it);
          personLikeProfileUri = uicomponent.getUserProfileUri(it);
          personLikeAvatarImageSource = uicomponent.getUserAvatarImageSource(it);
          if (!personLikeAvatarImageSource) {
                    personLikeAvatarImageSource = "/social-resources/skin/ShareImages/activity/
AvatarPeople.gif";
          }
          %>
            <a class="AvatarPeopleBG" title="$personLikeFullName" href="$personLikeProfileUri">
                        <img width="47px" height="47px" src="$personLikeAvatarImageSource"
 alt="$personLikeFullName" title="$personLikeFullName" />
            </a>
         <% })%>
        </div>
        <div class="ClearLeft">
          <span></span>
        </div>
      </div>
    </div>
  </div>

  <div class="CommentListInfo">
  <% if (uicomponent.commentListToggleable()) {
  def showAllCommentsAction = uicomponent.event("SetCommentListStatus", "all");
  def hideAllCommentsAction = uicomponent.event("SetCommentListStatus", "none");
  %>
    <div class="CommentBlock">
      <div class="CommentContent">
        <div class="CommentBorder">
                                    <% if (commentListStatus.getStatus().equals("latest") ||
 commentListStatus.getStatus().equals("none")) { %>
          <a onclick="$showAllCommentsAction" href="#show-all-comments">
            $labelShowAllComments
          </a>
        <% } else if (commentListStatus.getStatus().equals("all")) { %>
```

```
          <a onclick="$hideAllCommentsAction" href="#hide-all-comments">
            $labelHideAllComments
          </a>
        <% } %>
        </div>
      </div>
    </div>
  <% } %>
  </div>
 <% if (allComments.size() > 0) { %>
  <div class="DownIconCM"><span></span></div>
 <% }%>

 <%
  def commenterFullName, commenterProfileUri, commenterImageSource, commentMessage,
commentPostedTime;
 def first = true, commentContentClass;
 commentList.each({
  if (first & !uicomponent.commentListToggleable()) {
    commentContentClass = "CommentContent";
    first = false;
  } else {
    commentContentClass = "";
  }
  commenterFullName = uicomponent.getUserFullName(it.userId);
  commenterProfileUri = uicomponent.getUserProfileUri(it.userId);
  commenterImageSource = uicomponent.getUserAvatarImageSource(it.userId);
  if (!commenterImageSource) {
    commenterImageSource = "/social-resources/skin/ShareImages/activity/AvatarPeople.gif";
  }
  commentMessage = it.title;
  commentPostedTime = uicomponent.getPostedTimeString(it.postedTime);
 %>
  <div id="CommentBlock${activity.id}" class="CommentBox clearfix">
    <a class="AvatarCM" title="$commenterFullName" href="$commenterProfileUri">
              <img src="$commenterImageSource" alt="$commenterFullName" height="36px"
width="38px">
    </a>
    <div class="ContentBox">
      <div class="Content">
                                                  <a href="$commenterProfileUri"><span
class="Commenter">$commenterFullName<span></a><br />
        $commentMessage
        <br/>
```

```
      </div>
      <div class="LinkCM">
        <span class="DateTime">$commentPostedTime</span>
      </div>
    </div>
  <%
    if (uicomponent.isCommentDeletable(it.userId)) {
  %>
        <div onclick="<%= uicomponent.event("DeleteComment", uicomponent.id, it.id); %>"
class="CloseCMContentHilight"><span></span></div>
  <% } %>
  </div>
 <% }) %>

                        <div class="CommentBox $commentFormBlockClass clearfix"
id="CommentFormBlock${activity.id}">
    <% uicomponent.renderChild(UIFormTextAreaInput.class); %>
                    <input type="button" onclick="<%= uicomponent.event("PostComment")
          %>" value="$labelComment" class="CommentButton DisplayNone"
id="CommentButton${activity.id}" />
  </div>

 </div>
 <% uiform.end() %>
</div>
<% } else { %> <!-- activity deleted -->
<div class="UIActivity Deleted">$labelActivityHasBeenDeleted</div>
<% }%>
```

And you should make needed modifications for this template:

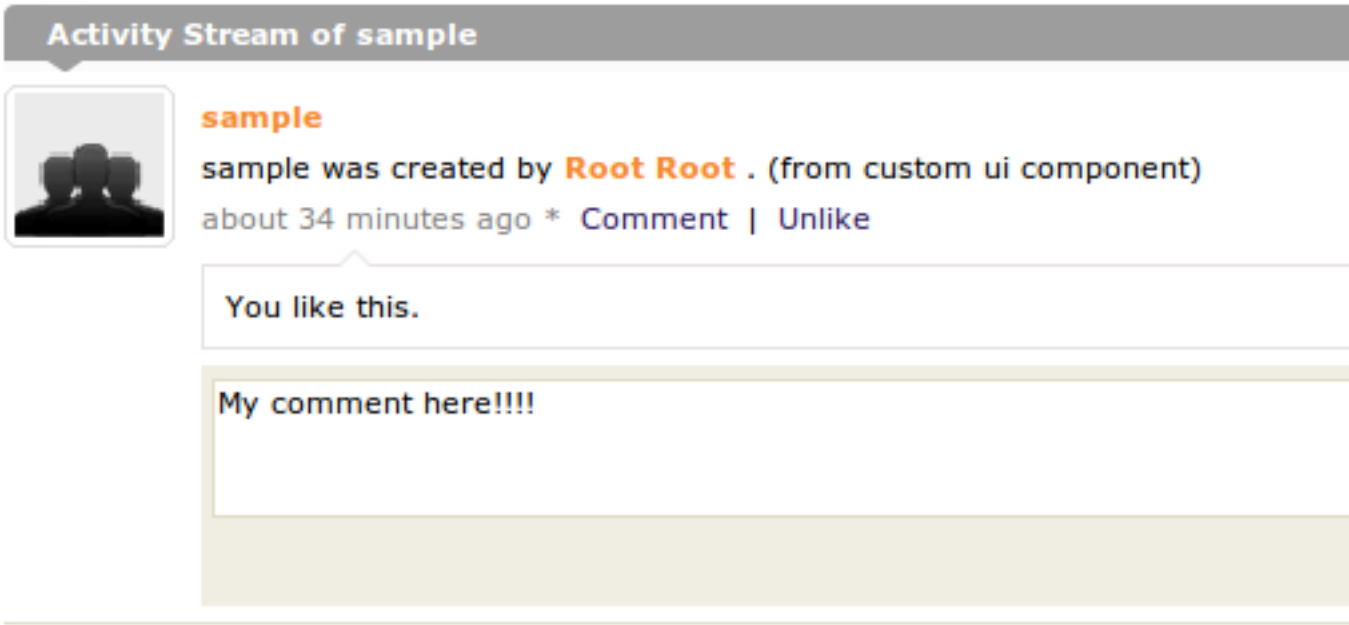
```
<div class="Content">
 $activityContentTitle (from custom UI component)<br>
</div>
```

7. Reconfigure the *configuration.xml* file:

```
<configuration xmlns="http://www.exoplaform.org/xml/ns/kernel_1_1.xsd" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.exoplaform.org/xml/
ns/kernel_1_1.xsd http://www.exoplaform.org/xml/ns/kernel_1_1.xsd">
```

```
  <external-component-plugins>
    <target-component>org.exoplatform.webui.ext.UIExtensionManager</target-component>
    <component-plugin>
      <name>add.action</name>
      <set-method>registerUIExtensionPlugin</set-method>
      <type>org.exoplatform.webui.ext.UIExtensionPlugin</type>
      <init-params>
        <object-param>
          <name>Look And Feel Space Activity</name>
          <object type="org.exoplatform.social.webui.activity.UIActivityExtension">
            <field name="type">
              <string>org.exoplatform.social.webui.activity.BaseUIActivity</string>
            </field>
            <field name="name">
              <string>exosocial:spaces</string>
            </field>
            <field name="component">
                <string>org.exoplatform.social.samples.activityplugin.UISpaceLookAndFeelActivity</
string>
            </field>
            <field name="activityBuiderClass">
                <string>org.exoplatform.social.samples.activityplugin.SimpleSpaceUIActivityBuilder</
string>
            </field>
          </object>
        </object-param>
      </init-params>
    </component-plugin>
  </external-component-plugins>
</configuration>
```

8. Rebuild the sample project, copy the **.jar** file to tomcat/lib. Run the server again and see the result:



Note

Currently, you have to copy and paste in the template file. By this way, you have full control of the UI but it is not a good way when there are changes in UIDefaultActivity. This will be improved soon so that no copy/paste is needed.

What is ActivityBuilder?

ActivityBuilder is one class which is used to get values of *ExoSocialActivity* to set to UIActivity for displaying. eXo Social provides the BaseUIActivityBuilder class for developers to extend and customize their own activity builder easily and properly.

For example, to write your own UILinkActivityBuilder, you just need to extend BaseUIActivityBuilder and then customize attributes and behaviors of the activity builder as below.

```
public class UILinkActivityBuilder extends BaseUIActivityBuilder {
  private static final Log LOG = ExoLogger.getLogger(UILinkActivityBuilder.class);
  @Override
  protected void extendUIActivity(BaseUIActivity uiActivity, ExoSocialActivity activity) {
    UILinkActivity uiLinkActivity = (UILinkActivity) uiActivity;
    Map<String, String> templateParams = activity.getTemplateParams();
    uiLinkActivity.setLinkSource(templateParams.get(UILinkActivityComposer.LINK_PARAM));
    uiLinkActivity.setLinkTitle(templateParams.get(UILinkActivityComposer.TITLE_PARAM));
    uiLinkActivity.setLinkImage(templateParams.get(UILinkActivityComposer.IMAGE_PARAM));

 uiLinkActivity.setLinkDescription(templateParams.get(UILinkActivityComposer.DESCRIPTION_PARAM));
```

```
 uiLinkActivity.setLinkComment(templateParams.get(UILinkActivityComposer.COMMENT_PARAM));
  }
}
```

Note

To learn more about ActivityBuilder, refer to the *BaseUIActivity class.* [http://svn.exoplatform.org/projects/social/trunk/component/webui/src/main/java/org/exoplatform/social/webui/activity/BaseUIActivity.java]

Create a composer extension for composing activity on the UI composer and display it on the activity stream

The UIActivityComposer is an extended class of UIContainer that is used to display inputs for users to create their own activities. To write your own activity composer, it is recommended that you use the UIActivityComposer available in eXo Social.

For example, to create an input component for inserting video links into your activity, do the following steps:

1. Write UIVideoActivityComposer which extends UIActivityComposer. The UIActivityComposer allows you to input extended activities (for example, adding videos, links or documents) on the UI composer.

```
package org.exoplatform.social.plugin.videolink;

import org.exoplatform.social.plugin.videolink.util.VideoEmbedTool;
@ComponentConfig(
        template = "classpath:groovy/social/plugin/videolink/UIVideoActivityComposer.gtmpl",
        events = {
                @EventConfig(listeners = UIVideoActivityComposer.SearchVideo.class),
               @EventConfig(listeners = UIVideoActivityComposer.SelectVideoFromResultList.class),
                @EventConfig(listeners = UIVideoActivityComposer.AttachActionListener.class),
                                                                @EventConfig(listeners =
 UIVideoActivityComposer.ChangeLinkContentActionListener.class),
                @EventConfig(listeners = UIActivityComposer.CloseActionListener.class),
                @EventConfig(listeners = UIActivityComposer.SubmitContentActionListener.class),
                @EventConfig(listeners = UIActivityComposer.ActivateActionListener.class)
        }
)

public class UIVideoActivityComposer extends UIActivityComposer {
```

```
public static final String LINK_PARAM = "link";
public static final String IMAGE_PARAM = "image";
public static final String TITLE_PARAM = "title";
public static final String HTML_PARAM = "htmlembed";
public static final String COMMENT_PARAM = "comment";

private static final String HTTP = "http://";
private static final String HTTPS = "https://";
private JSONObject videoJson;
private boolean linkInfoDisplayed_ = false;
private Map<String, String> templateParams;

/**
 * The constructor.
 */
public UIVideoActivityComposer() {
  setReadyForPostingActivity(false);
  addChild(new UIFormStringInput("InputLink", "InputLink", null));
}
/**
 * Set the link info to be displayed.
 *
 * @param displayed
 */
public void setLinkInfoDisplayed(boolean displayed) {
  linkInfoDisplayed_ = displayed;
}

/**
 * Set the template params.
 *
 * @param templateParams
 */
public void setTemplateParams(Map<String, String> templateParams) {
  this.templateParams = templateParams;
}

/**
 * Get the template params.
 */
public Map<String, String> getTemplateParams() {
  return templateParams;
}
```

```
/**
 * Clear the video json.
 */
public void clearVideoJson() {
  videoJson = null;
}

/**
 * Get the video json.
 */
public JSONObject getVideoJson() {
  return videoJson;
}

/**
 * Set the link.

 * @param url
 * @throws Exception
 */
private void setLink(String url) throws Exception {
  if (!(url.contains(HTTP) || url.contains(HTTPS))) {
    url = HTTP + url;
  }

  videoJson = VideoEmbedTool.getoembedData(url);
  templateParams = new HashMap<String, String>();
  templateParams.put(LINK_PARAM, url);
                                                    templateParams.put(TITLE_PARAM,
videoJson.getString(VideoEmbedTool.OEMBED_TITLE));
                                                    templateParams.put(HTML_PARAM,
videoJson.getString(VideoEmbedTool.OEMBED_HTML));
  setLinkInfoDisplayed(true);
}

static public class AttachActionListener extends EventListener<UIVideoActivityComposer> {

  @Override
  public void execute(Event<UIVideoActivityComposer> event) throws Exception {
    WebuiRequestContext requestContext = event.getRequestContext();
    UIVideoActivityComposer uiComposerLinkExtension = event.getSource();
    String url = requestContext.getRequestParameter(OBJECTID);
    try {
```

```
      uiComposerLinkExtension.setLink(url.trim());
    } catch (Exception e) {
      uiComposerLinkExtension.setReadyForPostingActivity(false);
      return;
    }
    requestContext.addUIComponentToUpdateByAjax(uiComposerLinkExtension);
    event.getSource().setReadyForPostingActivity(true);
  }
 }

        static public class ChangeLinkContentActionListener extends
EventListener<UIVideoActivityComposer> {
  @Override
  public void execute(Event<UIVideoActivityComposer> event) throws Exception {
    WebuiRequestContext requestContext = event.getRequestContext();
    UIVideoActivityComposer uiComposerLinkExtension = event.getSource();

    Map<String, String> tempParams = new HashMap<String, String>();

    uiComposerLinkExtension.setTemplateParams(tempParams);
    requestContext.addUIComponentToUpdateByAjax(uiComposerLinkExtension);
    UIComponent uiParent = uiComposerLinkExtension.getParent();
    if (uiParent != null) {
      uiParent.broadcast(event, event.getExecutionPhase());
    }
  }
 }

        public static class SelectVideoFromResultList extends
EventListener<UIVideoActivityComposer>{
  @Override
  public void execute(Event<UIVideoActivityComposer> event) throws Exception {
    WebuiRequestContext requestContext = event.getRequestContext();
    UIVideoActivityComposer uiComposerLinkExtension = event.getSource();

  }
 }

 public static class SearchVideo extends EventListener<UIVideoActivityComposer>{

  @Override
  public void execute(Event<UIVideoActivityComposer> event) throws Exception {
    WebuiRequestContext requestContext = event.getRequestContext();
    UIVideoActivityComposer uiComposerLinkExtension = event.getSource();
```

```
  }
 }

 @Override
 public void onPostActivity(PostContext postContext, UIComponent source,
                 WebuiRequestContext requestContext, String postedMessage) throws Exception {

  templateParams.put(COMMENT_PARAM, postedMessage);
  setTemplateParams(templateParams);
  if (templateParams.size() == 0) {
                                                                uiApplication.addMessage(new
ApplicationMessage("UIComposer.msg.error.Empty_Message",
          null,
          ApplicationMessage.WARNING));
    return;
  }
  String title = "Shared a video: <a href={{{"}}}${" + LINK_PARAM + "}{{{"}}}>${" + TITLE_PARAM
+ "} </a>";
  ExoSocialActivity activity = new ExoSocialActivityImpl(userIdentity.getId(),
        UIVideoActivity.ACTIVITY_TYPE,
        title,
        null);
  activity.setTemplateParams(templateParams);

  if (postContext == UIComposer.PostContext.SPACE) {

                                        UIActivitiesContainer activitiesContainer =
uiDisplaySpaceActivities.getActivitiesLoader().getActivitiesContainer();
    activitiesContainer.addActivity(activity);
    requestContext.addUIComponentToUpdateByAjax(activitiesContainer);
    requestContext.addUIComponentToUpdateByAjax(uiComposer);
  } else if (postContext == PostContext.USER) {
                UIUserActivitiesDisplay uiUserActivitiesDisplay = (UIUserActivitiesDisplay)
getActivityDisplay();
    String ownerName = uiUserActivitiesDisplay.getOwnerName();
                                                Identity ownerIdentity =
identityManager.getOrCreateIdentity(OrganizationIdentityProvider.NAME,
          ownerName, false);

    activityManager.saveActivity(ownerIdentity, activity);
```

```
                                    if (uiUserActivitiesDisplay.getSelectedDisplayMode() ==
 UIUserActivitiesDisplay.DisplayMode.MY_STATUS) {
                                            UIActivitiesContainer activitiesContainer =
 uiUserActivitiesDisplay.getActivitiesLoader().getActivitiesContainer();
       if (activitiesContainer.getChildren().size() == 1) {

 uiUserActivitiesDisplay.setSelectedDisplayMode(UIUserActivitiesDisplay.DisplayMode.MY_STATUS);
       } else {
         activitiesContainer.addActivity(activity);
         requestContext.addUIComponentToUpdateByAjax(activitiesContainer);
         requestContext.addUIComponentToUpdateByAjax(uiComposer);
       }
     } else{

 uiUserActivitiesDisplay.setSelectedDisplayMode(UIUserActivitiesDisplay.DisplayMode.MY_STATUS);
     }
   }
 }
}
```

2. Use the BaseUIActivity class to write and customize the UIActivity display as below:

```
package org.exoplatform.social.plugin.videolink;
import org.exoplatform.social.webui.activity.BaseUIActivity;
import org.exoplatform.webui.config.annotation.ComponentConfig;
import org.exoplatform.webui.core.lifecycle.UIFormLifecycle;
import org.exoplatform.webui.config.annotation.EventConfig;
@ComponentConfig(lifecycle = UIFormLifecycle.class, template = "classpath:groovy/social/
plugin/videolink/UIVideoActivity.gtmpl", events = {
  @EventConfig(listeners = BaseUIActivity.ToggleDisplayLikesActionListener.class),
  @EventConfig(listeners = BaseUIActivity.ToggleDisplayCommentFormActionListener.class),
  @EventConfig(listeners = BaseUIActivity.LikeActivityActionListener.class),
  @EventConfig(listeners = BaseUIActivity.SetCommentListStatusActionListener.class),
  @EventConfig(listeners = BaseUIActivity.PostCommentActionListener.class),
     @EventConfig(listeners = BaseUIActivity.DeleteActivityActionListener.class, confirm =
 "UIActivity.msg.Are_You_Sure_To_Delete_This_Activity"),
    @EventConfig(listeners = BaseUIActivity.DeleteCommentActionListener.class, confirm =
 "UIActivity.msg.Are_You_Sure_To_Delete_This_Comment")
 }
)
public class UIVideoActivity extends BaseUIActivity {
  public static final String ACTIVITY_TYPE = "VIDEO_ACTIVITY";
```

```
private String linkSource = "";
private String linkTitle = "";
private String linkHTML = "";
private String linkComment = "";

/**
 * Get the link comment.
 */
public String getLinkComment() {
  return linkComment;
}

/**
 * Set the link comment.
 *
 * @param linkComment
 */
public void setLinkComment(String linkComment) {
  this.linkComment = linkComment;
}

/**
 * Get the link html.
 */
public String getLinkHTML() {
  return linkHTML;
}

/**
 * Set the link html.
 *
 * @param linkHTML
 */
public void setLinkHTML(String linkHTML) {
  this.linkHTML = linkHTML;
}

/**
 * Get the link source.
 */
public String getLinkSource() {
  return linkSource;
}
```

```
  /**
   * Set the link source.
   *
   * @param linkSource
   */
  public void setLinkSource(String linkSource) {
    this.linkSource = linkSource;
  }

  /**
   * Get the link title.
   */
  public String getLinkTitle() {
    return linkTitle;
  }

  /**
   * Set the link title.
   *
   * @param linkTitle
   */
  public void setLinkTitle(String linkTitle) {
    this.linkTitle = linkTitle;
  }
}
```

3. Use the *UIVideoActivityBuilder* class to get values of *ExoSocialActivity* that are set to UIVideoActivity for displaying.

```
package org.exoplatform.social.plugin.videolink;
import java.util.Map;
import org.exoplatform.social.core.activity.model.ExoSocialActivity;

public class UIVideoActivityBuilder extends BaseUIActivityBuilder {
  private static final Log LOG = ExoLogger.getLogger(UIVideoActivityBuilder.class);
  @Override
  protected void extendUIActivity(BaseUIActivity uiActivity, ExoSocialActivity activity) {
  UIVideoActivity uiVideoActivity = (UIVideoActivity) uiActivity;
  Map<String, String> templateParams = activity.getTemplateParams();
   uiVideoActivity.setLinkSource(templateParams.get(UIVideoActivityComposer.LINK_PARAM));
    uiVideoActivity.setLinkTitle(templateParams.get(UIVideoActivityComposer.TITLE_PARAM));
```

```
 uiVideoActivity.setLinkImage(templateParams.get(UIVideoActivityComposer.IMAGE_PARAM));
  uiVideoActivity.setLinkHTML(templateParams.get(UIVideoActivityComposer.HTML_PARAM));

 uiVideoActivity.setLinkComment(templateParams.get(UIVideoActivityComposer.COMMENT_PARAM));
  }
}
```

Note

You can check out the *source code* [https://github.com/exosocial/exo.social.extras.youtube-composer-plugin] to get more details.

Overridable Components

There are 2 components in eXo Social that can be overridden: Space Application Handler & Space Service.

- **Space Application Handler**

```
<component>
  <key>org.exoplatform.social.core.space.spi.SpaceApplicationHandler</key>
  <type>org.exoplatform.social.core.space.impl.DefaultSpaceApplicationHandler</type>
</component>
```

- **Space Service**

```
<component>
  <key>org.exoplatform.social.core.space.spi.SpaceService</key>
  <type>org.exoplatform.social.core.space.impl.SpaceServiceImpl</type>
  <init-params>
    <!-- Configure the applications to install in a space -->
    <values-param>
      <name>space.homeNodeApp</name>
      <value>SpaceActivityStreamPortlet</value>
    </values-param>
    <!-- Configure removable application or not <value>Application:removable</value> -->
    <values-param>
      <name>space.apps</name>
      <value>DashboardPortlet:true</value>
      <value>SpaceSettingPortlet:false</value>
```

```
      <value>MembersPortlet:true</value>
    </values-param>
  </init-params>
</component>
```

Public Java APIs

ActivityManager

Method	Param	Return	Description
saveActivity(Identity owner, ExoSocialActivity activity) throws ActivityStorageException	`owner` - the owner of activity stream, `activity` - the activity which needs to be saved	ExoSocialActivity	Save an activity to the stream of an owner. Note that the Activity.userId will be set to the owner's identity if it has not been already set.
getActivity(String activityId) throws ActivityStorageException	`activityId` - the id of activity	ExoSocialActivity	Get an activity by its id.
deleteActivity(String activityId) throws ActivityStorageException	`activityId` - the id of activity	void	Delete an activity by its id.
deleteActivity(ExoSocialActivity activity) throws ActivityStorageException	`activity`	void	Delete a stored activity (id != null). (Since 1.1.1).
deleteComment(String activityId, String commentId) throws ActivityStorageException	`activityId` - the id of activity, `commentId` - the id of comment	void	Delete a comment by its id.
getActivities(Identity identity) throws ActivityStorageException	`identity`	List<ExoSocialActivity>	Get the latest activities by an identity with the default limit of 20 latest activities.
getActivities(Identity identity, long start, long limit) throws ActivityStorageException	`identity`, `start`, `limit`	List<ExoSocialActivity>	Get the latest activities by an identity, specifying **start** that is an offset index and **limit**.
getActivitiesOfConnections(Identity ownerIdentity) throws ActivityStorageException	`ownerIdentity`	List<ExoSocialActivity>	Get activities of connections from an identity. The activities are returned as a list that is sorted

Method	Param	Return	Description
			descending by activity posted time. (Since 1.1.1).
getActivitiesOfConnections(Identity ownerIdentity, int offset, int limit) throws ActivityStorageException;	`ownerIdentity, offset, limit`	List<ExoSocialActivity>	Get the activities of connections from an identity by specifying offset and limit. The activities are returned as a list that is sorted starting from the most recent activity.(Since 1.2.0-GA).
getActivitiesOfUserSpaces(Identity ownerIdentity)	`ownerIdentity`	List<ExoSocialActivity>	Get the activities from all spaces of a user. By default, the activity list is composed of all spaces' activities. Each activity list of the space contains maximum 20 activities and are sorted by time. (Since 1.1.1).
getActivityFeed(Identity identity) throws ActivityStorageException	`identity`	List<ExoSocialActivity>	Get the activity feed of an identity. This feed is the combination of all the activities of his own activities, his connections' activities and spaces' activities which are returned as a list that is sorted starting from the most recent activity.(Since 1.1.2).
saveActivity(ExoSocialActivity activity) throws ActivityStorageException	`activity` - the activity to save	ExoSocialActivity	Save an activity into the stream for the activity's userId. The userId must be set and this field is used to indicate the owner stream.

Method	Param	Return	Description
saveComment(ExoSocialActivity activity, ExoSocialActivity comment) throws ActivityStorageException	`activity, comment`	void	Save a new comment or updates an existing comment that is an instance of activity with mandatory fields: userId, title.
saveLike(ExoSocialActivity activity, Identity identity) throws ActivityStorageException	`activity, identity`	void	Save an identity who likes an activity.
removeLike(ExoSocialActivity activity, Identity identity) throws ActivityStorageException	`activity, identity` - a user who dislikes an activity	void	Remove an indentity who likes an activity, if this activity is liked, it will be removed.
getComments(ExoSocialActivity activity) throws ActivityStorageException	activity	List<ExoSocialActivity>	Get the comment list of an activity.
recordActivity(Identity owner, String type, String title) throws ActivityStorageException	`owner, type, title`	ExoSocialActivity	Record an activity. (Since 1.2.0-GA).
recordActivity(Identity owner, ExoSocialActivity activity) throws Exception	`owner, activity`	ExoSocialActivity	Save an activity. You should use ActivityManager#saveActivity(org.exoplatf org.exoplatform.social.core.activity.model. instead. It will be removed in eXo Social 1.3.x.
recordActivity(Identity owner, String type, String title, String body) throws ActivityStorageException	`owner` - the owner of the target stream for this activity, `type` - the type of an activity which will be used to render a custom UI, `title` - the title, `body` - the body	ExoSocialActivity	Record an activity.
addProcessor(ActivityProcessor processor)	`processor`	void	Add a new processor.
addProcessorPlugin(BaseActivityProcessorPlugin plugin)	`plugin`	void	Add a new processor plugin.

Method	Param	Return	Description
getActivitiesCount(Identity owner) throws ActivityStorageException	`owner`	int	Get the number of activities from a stream owner.
processActivitiy(ExoSocialActivity activity)	`activity`	void	Pass an activity through the chain of processors.

IdentityManager

Method	Param	Return	Description
registerIdentityProviders(IdentityProviderPlugin plugin)	`plugin`	void	Register one or more IdentityProvider through an IdentityProviderPlugin.
getIdentity(String id)	`id` can be a social GlobalId or a raw identity such as in Identity.getId()	Identity - null if nothing is found, or the Identity object	Get the identity by ID and loads his profile.
getIdentity(String id, boolean loadProfile)	`id` can be a social GlobalId or a raw identity such as in Identity.getId(), `loadProfile` - the value is true if the profile is loaded and false if not loaded	null if nothing is found, or the Identity object	Get the identity by loading id of the profile optionally.
deleteIdentity(Identity identity)	`identiy`	void	Delete an identity.
addIdentityProvider(IdentityProvider idProvider)	`identityProvider` - the id of provider	void	Add the identity provider.
getOrCreateIdentity(String providerId, String remoteId)	`providerId` - the id of provider, `remoteId` - the remote id	Identity	Get the identity by remoteId. If the provider can not find any identity by remoteId, the return value is null. If no identity found by identity provider and that identity is still stored on JCR, the stored identity will be

Method	Param	Return	Description
			deleted and the return value is null.
getOrCreateIdentity(String providerId, String remoteId, boolean loadProfile)	`providerId` - referring to the name of the Identity provider, `remoteId` - the identifier that identify the identity in the specific identity provider, `loadProfile` - true when the profile is loaded	null if nothing is found, or the Identity object improves the performance by specifying what needs to be loaded	This function returns an Identity object that is specific to a special type. For example, if the type is Linked'In, the identifier will be the URL of profile or if it is a CS contact manager, it will be the UID of the contact. A new identity is created if it does not exist.
getIdentitiesByProfileFilter(String providerId, ProfileFilter profileFilter) throws Exception	`providerId` - the id of provider, `profileFilter` - the filter of provider	Identity	Get the identities by a profile filter.
getIdentitiesByProfileFilter(String providerId, ProfileFilter profileFilter, long offset, long limit) throws Exception	`providerId`, `profileFilter`, `offset`, `limit`	List<Identity>	Get the identities by a profile filter.
getIdentitiesByProfileFilter(ProfileFilter profileFilter) throws Exception	`profileFilter` - the profile filter	List<Identity>	Get the identities by a profile filter.
getIdentitiesByProfileFilter(ProfileFilter profileFilter, long offset, long limit) throws Exception	`profileFilter`, `offset`, `limit`	List<Identity>	Get the identities by a profile filter.
getIdentitiesFilterByAlphabet(String providerId, ProfileFilter profileFilter) throws Exception	`providerId` - the id of provider, `profileFilter` - the profile filter	List<Identity>	Get the identities filter by alphabet.
getIdentitiesFilterByAlphabet(String providerId, ProfileFilter profileFilter, long	`providerId`, `profileFilter`, `offset`, `limit`	List<Identity>	Get the identities filter by alphabet with offset and limit.

Method	Param	Return	Description
offset,long limit) throws Exception			
getIdentitiesFilterByAlphabet(ProfileFilter profileFilter) throws Exception	`profileFilter` - the profile filter	List<Identity>	Get the identities filter by alphabet.
getIdentity(String providerId, String remoteId, boolean loadProfile)	`providerId`, `remoteId`, `loadProfile`	Identity	Get the identity.
getIdentitiesCount(String providerId)	`providerId`	long	Get the number of indentities.
identityExisted(String providerId, String remoteId)	`providerId`, `remoteId`	boolean	Check if the identity is already existed or not.
saveIdentity(Identity identity)	`identity` - the identity	void	Save the identity.
saveProfile(Profile profile)	`profile`	void	Save a profile.
addOrModifyProfileProperties(Profile profile) throws Exception	`profile`	void	Add or modify properties of profile. Profile parameter is a lightweight that contains only the property that you want to add or modify. NOTE: The method will not delete the properties of an old profile when the param profile does not have those keys.
updateAvatar(Profile p)	`profile`	void	Update the avatar.
updateBasicInfo(Profile p) throws Exception	`profile`	void	Update the basic infomation.
updateContactSection(Profile p) throws Exception	`profile`	void	Update the contact section of the profile.
updateExperienceSection(Profile p)throws Exception	`profile`	void	Update the experience section of the profile.

Method	Param	Return	Description
updateHeaderSection(Profile p) throws Exception	`profile`	void	Update the header section of the profile.
getIdentities(String providerId) throws Exception	`providerId` - the id of provider	List<Identity>	Get the identities by the provider id.
getIdentities(String providerId, boolean loadProfile)	`providerId` - the id of provider, `loadProfile` - the loaded profile.	List<Identity>	Get the identities by the provider id. If loadProvider is true, loading the profie will be perfomed.
getConnections(Identity ownerIdentity) throws Exception	`ownerIdentity`	List<Identity>	Get connections of an identity. (Since 1.1.1).
getIdentityStorage()	`N/A`	IdentityStorage	Get the identity storage.
getStorage()	`N/A`	IdentityStorage	Get the storage. Deprecated: should use method getIdentityStorage().
registerProfileListener(ProfileListener listener)	`listener`	void	Register the profile listener.
unregisterProfileListener(ProfileListener listener)	`listener`	void	Unregister the profile listener.
addProfileListener(ProfileListenerPlugin plugin)	`plugin`	void	Register a profile listener component plug-in.

RelationshipManager

Method	Param	Return	Description
getRelationshipById(String id) throws Exception	`id`	Relationship	Get the relationship by id. You should use get(String) instead. It will be removed at 1.2.x.
invite(Identity sender, Identity receiver) throws RelationshipStorageException	`sender` `receiver`	Relationship	Create a connection invitation between two identities.
		void	Save a relationship.

Method	Param	Return	Description
saveRelationship(Relationship relationship) throws RelationshipStorageException	`relationship` - a relationship		
confirm(Relationship relationship) throws RelationshipStorageException	`relationship` - a relationship	void	Mark a relationship as confirmed.
deny(Relationship relationship) throws RelationshipStorageException	`relationship` - a relationship	void	Deny a relationship.
remove(Relationship relationship) throws RelationshipStorageException	`relationship` - a relationship	void	Remove a relationship.
ignore(Relationship relationship) throws RelationshipStorageException	`relationship` - a relationship	void	Mark a relationship as ignored
getPendingRelationships(Identity sender) throws Exception	`identity` - an identity	List<Relationship>	Get all the pending relationship of sender.
getPendingRelationships(Identity sender, List<Identity> identities) throws Exception	`identity` - an identity, `identities` - a list of identity	List<Relationship>	Get pending relationships of sender that match with identities.
getRequireValidationRelationships(Identity receiver) throws Exception	`identity` - an identity	List<Relationship>	Get list of required validation relationship of receiver.
getRequireValidationRelationships(Identity receiver, List<Identity> identities)	`identity` - an identity, `identities` - a list of identity	List<Relationship>	Get list of required validation relationship of receiver that match with identities.
getConfirmedRelationships(Identity identity)	`identity` - an identity	List<Relationship>	Get list of confirmed relationship of identity.
getConfirmedRelationships(Identity identity, List<Identity> identities)	`identity` - an identity, `identities` - a list of identity	List<Relationship>	Get list of confirmed relationship of identity that match with identities.
getAllRelationships(Identity identity)	`identity` - an identity	List<Relationship>	Return all the relationship of a given identity with other identity.
getAllRelationships(Identity identity, List<Identity> identities)	`identity` - an identity, `identities` - a list of identity	List<Relationship>	Return all the relationship of a given identity with other identity in identities.

Method	Param	Return	Description
getAllRelationships(Identity identity)	`identity` - an identity	List<Relationship>	Return all the relationship of a given identity with other identity.
getAllRelationships(Identity identity, Relationship.Type type, List<Identity> identities)	`identity` - an identity, `type` - a Relationship.Type, `identities` - a list of identity <Relationship>		Return all the relationship of a given identity with other identity in identities in type.
getRelationship(Identity identity1, Identity identity2)	`identity1` and `identity2` - identities	Relationship	Get the relationship of two identities.

SpaceService

Method	Param	Return	Description
getSpaceByDisplayName(String spaceDisplayName)	`spaceDisplayName`	Space	Get a space whose display name matches the string input. (Since 1.2.0-GA).
getSpaceByPrettyName(String spaceName)	`spaceName`	Space	Get a space whose pretty name matches the string input. (Since 1.2.0-GA).
getSpaceByGroupId(String groupId)	`groupId`	Space	Get a space that has group Id matching the string input.
getSpaceById(String spaceId)	`spaceId`	Space	Get a space by its Id.
getSpaceByUrl(String spaceUrl)	`spaceUrl`	Space	Get a space whose URL matches the string input.
getAllSpaces()	`N/A`	ListAccess<Space>	Get a list of spaces with the type of a space list access. (Since 1.3.0-GA).
getAllSpacesWithListAccess()	`N/A`	ListAccess<Space>	Get a list of spaces with the type of a space list access. (Since 1.2.0-GA).

Method	Param	Return	Description
getAllSpacesByFilter(SpaceFilter spaceFilter)	`spaceFilter`	ListAccess<Space>	Get a list of spaces with the type of a space list access. These spaces matches the space filter. (Since 1.2.0-GA).
getMemberSpaces(String userId)	`userId`	ListAccess<Space>	Get a list of spaces with the type of list access that contains all the spaces in which a user has the "member" role. (Since 1.2.0-GA).
getMemberSpacesByFilter(String userId, SpaceFilter spaceFilter)	`userId`, `spaceFilter`	ListAccess<Space>	Get a list of spaces with the type of a space list access. The list contains the spaces which a user has the "member" role and match the provided space filter. (Since 1.2.0-GA).
getAccessibleSpaces(String userId)	`userId`	ListAccess<Space>	Get a list of spaces with the type of a space list access. The list contains all the spaces that a user has the access permission.(Since 1.3.0-GA).
getAccessibleSpacesWithListAccess(String userId)	`userId`	ListAccess<Space>	Get a list of spaces with a space list access. The list contains all the spaces that a user has the access permission. (Since 1.2.0-GA).

Method	Param	Return	Description
getAccessibleSpacesByFilter(String userId, SpaceFilter spaceFilter)	`userId`, `spaceFilter`	ListAccess<Space>	Get a list of spaces with the type of a space list access. The list contains all the spaces that a user has the access permission and match the provided space filter. (Since 1.2.0-GA).
getSettingableSpaces(String userId)	`userId`	ListAccess<Space>	Get a list of spaces with the type of a space list access. The list contains all the spaces that a user has the setting permission. (Since 1.2.0-GA).
getSettingabledSpacesByFilter(String userId, SpaceFilter spaceFilter)	`userId`, `spaceFilter`	ListAccess<Space>	Get a list of spaces with the type of a space list access. The list contains all the spaces that a user has the setting permission and match the provided space filter. (Since 1.2.0-GA).
getInvitedSpaces(String userId)	`userId`	ListAccess<Space>	Get a list of spaces with the type of a space list access. The list contains all the spaces that a user is invited to join. (Since 1.3.0-GA).
getInvitedSpacesWithListAccess(String userId)	`userId`	ListAccess<Space>	Get a list of spaces with the type of a space list access. The list contains all the spaces that a user is invited to join. (Since 1.2.0-GA).

Method	Param	Return	Description
getInvitedSpacesByFilter(String userId, SpaceFilter spaceFilter)	`userId`, `spaceFilter`	ListAccess<Space>	Get a list of spaces with the type of a space list access. The list contains all the spaces that a user is invited to join and match the provided space filter. (Since 1.2.0-GA).
getPublicSpaces(String userId)	`userId`	ListAccess<Space>	Get a list of spaces with the type of a space list access. The list contains all the spaces that a user can request to join. (Since 1.3.0-GA).
getPublicSpacesWithListAccess(String userId)	`userId`	ListAccess<Space>	Get a list of spaces with the type of a space list access. The list contains all the spaces that a user can request to join. (Since 1.2.0-GA).
getPublicSpacesByFilter(String userId, SpaceFilter spaceFilter)	`userId`, `spaceFilter`	ListAccess<Space>	Get a list of spaces with the type of a space list access. The list contains all the spaces that a user can request to join and match the provided space filter. (Since 1.2.0-GA).
getPendingSpaces(String userId)	`userId`	ListAccess<Space>	Get a list of spaces with the type of a space list access. The list contains all the spaces that a user sent join-request to a space. (Since 1.3.0-GA).

Method	Param	Return	Description
getPendingSpacesWithListAccess(String userId)	`userId`	ListAccess<Space>	Get a list of spaces with the type of a space list access. The list contains all the spaces that a user sent a request for joining a space. (Since 1.2.0-GA).
getPendingSpacesByFilter(String userId, SpaceFilter spaceFilter)	`userId, spaceFilter`	ListAccess<Space>	Get a list of spaces with the type of a space list access. The list contains all the spaces that a user sent join-request to a space and match the provided space filter. (Since 1.2.0-GA).
createSpace(Space space, String creatorUserId)	`space, creatorUserId`	Space	Create a new space: create a group, its group navigation with pages for installing space applications.
updateSpace(Space existingSpace)	`existingSpace`	Space	Update information of a space. (Since 1.2.0-GA).
deleteSpace(Space space)	`space`	void	Delete a space. When deleting a space, all of its page navigation bars and its group will be deleted.
addPendingUser(Space space, String userId)	`space, userId`	void	Add a user to the pending list to request to join a space. (Since 1.2.0-GA).
removePendingUser(Space space, String userId)	`space, userId`	void	Remove a user from the pending list to request to join a space. (Since 1.2.0-GA).
isPendingUser(Space space, String userId)	`space, userId`	void	Check if a user is in the pending list to

Method	Param	Return	Description
			request to join a space or not. (Since 1.2.0-GA).
addInvitedUser(Space space, String userId)	`space, userId`	void	Add a user, who is invited to a space, to the invited list. (Since 1.2.0-GA).
removeInvitedUser(Space space, String userId)	`space, userId`	void	Remove a user, who is invited to a space, from the invited list. (Since 1.2.0-GA).
isInvitedUser(Space space, String userId)	`space, userId`	void	Check if a user invited to join a space is in the invited list or not. (Since 1.2.0-GA).
addMember(Space space, String userId)	`space, userId`	void	Add a user to a space. The user will get the "member" role in that space.
removeMember(Space space, String userId)	`space, userId`	void	Remove a member from a space.
isMember(Space space, String userId)	`space, userId`	boolean	Check whether a user is a space's member or not.
setManager(Space space, String userId, boolean isManager)	`space, userId, isManager`	void	Add a user to have the "manager" role in a space. If *isManager* is set to "true", a user will get the "manager" role. If false, that user will get the "member" role. (Since 1.2.0-GA).
isManager(Space space, String userId)	`space, userId`	void	Check if a user has the "manager" role in a space or not. (Since 1.2.0-GA).
isOnlyManager(Space space, String userId)	`space, userId`	boolean	Check if a user is the only one who has the "manager" role in a space. True if the user Id is the only one who

Method	Param	Return	Description
			has "manager" role in a space. Otherwise, return false. (Since 1.2.0-GA).
hasAccessPermission(Space space, String userId)	(Space space, `String userId`	boolean	Check if a user can access a space or not. If the user is root or the space's member, return true.
hasSettingPermission(Space space, String userId)	(Space space, `String userId`	boolean	Check if a user can have the setting permission to a space or not. If the user is root or the space's member, return true. (Since 1.2.0-GA).
registerSpaceListenerPlugin(SpaceListenerPlugin spaceListenerPlugin)	registerSpaceListenerPlugin(SpaceListenerPlugin	void	Register a space listener plugin to listen to space lifecyle events: creating, removing space, activating, deactiving, adding, removing application, promoting, joining, leaving, and revoking. (Since 1.2.0-GA).
unregisterSpaceListenerPlugin(SpaceListenerPlugin spaceListenerPlugin)	unregisterSpaceListenerPlugin(SpaceListenerPlugin	void	Unregister an existing space listener plugin. (Since 1.2.0-GA).
setSpaceApplicationConfigPlugin(SpaceApplicationConfigPlugin spaceApplicationConfigPlugin)	setSpaceApplicationConfigPlugin(SpaceApplicationConfigPlugin	void	Set a space application configuration plugin to configure the home and space applications. By configuring this, the space service will know how to create a new page node with title, URL, and portlet

Method	Param	Return	Description
			to use. (Since 1.2.0-GA).
getSpaceApplicationConfigPlugin()	N/A	SpaceApplicationConfigPlugin	Get the configuration of applications to be initialized when creating a new space. (Since 1.2.0-GA).
getAllSpaces() throws SpaceException	N/A	List<Space>	Get all spaces in eXo Social. You should use method getAllSpaceWithListAccess instead of getAllSpaces. It will be removed in eXo Social 1.3.x.
getSpaceByName(String spaceName) throws SpaceException	spaceName	Space	Get a space by its name. You should use SpaceService#getSpaceByPrettyName instead. It will be removed version 1.3.x.
getSpacesByFirstCharacterOfName(String firstCharacterOfName) throws SpaceException	firstCharacterOfName	List<Space>	Get all spaces whose name starting with the input character.
getSpacesBySearchCondition(String condition) throws Exception	condition	List<Space>	Get all spaces which has the name or the description that matches the input condition.
getSpaces(String userId) throws SpaceException	userId	List<Space>	Get spaces of a user in which that user is a member. You should use getMemberSpaces(String) instead. It will be removed in eXo Social 1.3.x

Method	Param	Return	Description
getAccessibleSpaces(String userId) throws SpaceException	`userId`	List<Space>	Get spaces of a user which that user has the access permission. You should use getAccessibleSpacesWithListAccess(Strin instead. It will be removed in eXo Social 1.3.x.
getEditableSpaces(String userId) throws SpaceException	`userId`	List<Space>	Get spaces of a user which that user has the edit permission. You should use getSettingableSpaces(String) instead. It will be removed in eXo Social 1.3.x.
getInvitedSpaces(String userId) throws SpaceException	`userId`	List<Space>	Get invited spaces of a user and that user can accept or deny the request. You should use getInvitedSpacesWithListAccess(String) instead. It will be removed in eXo Social 1.3.x.
getPublicSpaces(String userId) throws SpaceException	`userId - Id of user`	List<Space>	Get invited spaces of a user that can be accepted or denied by the user. You should use getPublicSpacesWithListAccess(String) instead. It will be removed in eXo Social 1.3.x.
getPendingSpaces(String userId) throws SpaceException	`userId`	List<Space>	Get pending spaces of a user and spaces which the user can revoke that request. You should use

Method	Param	Return	Description
			getPendingSpacesWithListAccess(String) instead. It will be removed in eXo Social 1.3.x.
createSpace(Space space, String creator, String invitedGroupId) throws SpaceException	`space, creator, invitedGroupId`	Space	Create a new space and invite all users from invitedGroupId to join this newly created space.
saveSpace(Space space, boolean isNew) throws SpaceException	`space, isNew`	void	Save a new space or update a space. You should use updateSpace(org.exoplatform.social.core.s instead. It will be removed in eXo Social 1.3.x.
deleteSpace(String spaceId) throws SpaceException	`spaceId`	void	Delete a space by its id. You should use deleteSpace(org.exoplatform.social.core.s instead. It will be removed in eXo Social 1.3.x.
initApp(Space space) throws SpaceException	`space`	void	It is just for compatibility. Deprecated: it will be removed in eXo Social 1.3.x.
initApps(Space space) throws SpaceException	`space`	void	It is just for compatibility. Deprecated: it will be removed in eXo Social 1.3.x.
deInitApps(Space space) throws SpaceException	`space`	void	It is just for compatibility. Deprecated: it will be removed in eXo Social 1.3.x.
addMember(String spaceId, String	`spaceId, userId`	void	Add a user to a space, the user will get the

Method	Param	Return	Description
userId) throws SpaceException			"member" role in a space. You should use addMember(org.exoplatform.social.core.sp String) instead. It will be removed in eXo Social 1.3.x.
removeMember(String spaceId, String userId) throws SpaceException	`spaceId, userId`	void	Remove a member from a space. You should use removeMember(org.exoplatform.social.cor String) instead. It will be removed in eXo Social 1.3.x.
getMembers(Space space) throws SpaceException	`space`	List<String>	Get a list of the space members from a space. You should use Space#getMembers() instead. It will be removed in eXo Social 1.3.x.
getMembers(String spaceId) throws SpaceException	`spaceId`	List<String>	Get a list of the space members from a space. You should use Space#getMembers() instead. It will be removed in eXo Social 1.3.x.
setLeader(Space space, String userId, boolean isLeader) throws SpaceException	`space, userId, isLeader`	void	Set a member of a space as a manager. You should use setManager(org.exoplatform.social.core.sp String, boolean) instead. It will be removed in eXo Social 1.3.x.
setLeader(String spaceId, String	`spaceId, userId, isLeader`	void	Set a member of a space as

Method	Param	Return	Description
userId, boolean isLeader) throws SpaceException			a manager. You should use setManager(org.exoplatform.social.core.sp String, boolean) instead. It will be removed in eXo Social 1.3.x.
isLeader(Space space, String userId) throws SpaceException	`space, userId`	boolean	Check whether a user is a space's leader or not. You should use isManager(org.exoplatform.social.core.spa String) instead. It will be removed in eXo Social 1.3.x.
isLeader(String spaceId, String userId) throws SpaceException	`spaceId, userId`	boolean	Check whether a user is a space's leader or not. You should use isManager(org.exoplatform.social.core.spa String) instead. It will be removed in eXo Social 1.3.x.
isOnlyLeader(Space space, String userId) throws SpaceException	`space, userId`	boolean	Check whether a user is the only leader of a space or not. You should use isOnlyManager(org.exoplatform.social.core String) instead. It will be removed in eXo Social 1.3.x.
isOnlyLeader(String spaceId, String userId) throws SpaceException	`spaceId, userId`	boolean	Check whether a user is the only leader of a space or not. You should use isOnlyManager(org.exoplatform.social.core String) instead. It will be removed in eXo Social 1.3.x.

Method	Param	Return	Description
isMember(String spaceId, String userId) throws SpaceException	`spaceId, userId,`	boolean	Check whether a user is a space's member or not. You should use isMember(org.exoplatform.social.core.spa String) instead. It will be removed in eXo Social 1.3.x.
hasAccessPermission(String spaceId, String userId) throws SpaceException	`spaceId, userId`	boolean	Check if a user can access a space or not. You should use hasAccessPermission(org.exoplatform.soc String) instead. It will be removed in eXo Social 1.3.x.
hasEditPermission(Space space, String userId) throws SpaceException	`space, userId`	Boolean	Check if a user can have the edit permission of a space or not. You should use hasSettingPermission(org.exoplatform.soc String) instead. It will be removed in eXo Social 1.3.x.
hasEditPermission(String spaceId, String userId) throws SpaceException	`spaceId, userId`	Boolean	Check if a user can have the edit permission of a space or not. You should use hasSettingPermission(org.exoplatform.soc String) instead. It will be removed in eXo Social 1.3.x.
isInvited(Space space, String userId) throws SpaceException	`space, userId`	Boolean	Check if a user is in the invited list of a space or not. You should use isInvitedUser(org.exoplatform.social.core.s String) instead. It will

Method	Param	Return	Description
			be removed in eXo Social 1.3.x.
isInvited(String spaceId, String userId) throws SpaceException	`spaceId, userId`	Boolean	Check if a user is in the invited list of a space or not. You should use isInvitedUser(org.exoplatform.social.core.s String) instead. It will be removed in eXo Social 1.3.x.
isPending(Space space, String userId) throws SpaceException	`space, userId`	Boolean	Check if a user is in the pending list of a space or not.You should use isPendingUser(org.exoplatform.social.core String) instead. It will be removed in eXo Social 1.3.x.
isPending(String spaceId, String userId) throws SpaceException	`spaceId, userId`	Boolean	Check if a user is in the pending list of a space or not. You should use isPendingUser(org.exoplatform.social.core String) instead. It will be removed in eXo Social 1.3.x.
installApplication(String spaceId, String appId) throws SpaceException	`spaceId, appId`	void	Install an application to a space.
installApplication(Space space, String appId) throws SpaceException	`space, appId`	void	Install an application to a space.
activateApplication(Space space, String appId) throws SpaceException	`space, appId`	void	Activate an installed application in a space.

Method	Param	Return	Description
activateApplication(String spaceId, String appId) throws SpaceException	`spaceId, appId`	void	Activate an installed application in a space.
deactivateApplication(Space space, String appId) throws SpaceException	`space, appId`	void	Deactivate an installed application in a space.
deactivateApplication(String spaceId, String appId) throws SpaceException	`spaceId, appId`	void	Deactivate an installed application in a space.
removeApplication(Space space, String appId, String appName) throws SpaceException	`space, appId, appName`	void	Remove an installed application from a space.
removeApplication(String spaceId, String appId, String appName) throws SpaceException	`space, appId, appName`	void	Remove an installed application from a space.
requestJoin(Space space, String userId) throws SpaceException	`space, userId`	void	Request users to join a space. The invited users are then added to the pending list of the space. You should use addPendingUser(org.exoplatform.social.co String) instead. It will be removed in eXo Social 1.3.x.
requestJoin(String spaceId, String userId) throws SpaceException	`spaceId, userId`	void	Request users to join a space. The invited users are then added to the pending list of the space.. You should use addPendingUser(org.exoplatform.social.co

Method	Param	Return	Description
			String) instead. It will be removed in eXo Social 1.3.x.
revokeRequestJoin(Space space, String userId) throws SpaceException	`space, userId`	void	Revoke a join request after users request to join a group and is in the pending status. You should use removePendingUser(org.exoplatform.socia String) instead. It will be removed in eXo Social 1.3.x.
revokeRequestJoin(String spaceId, String userId) throws SpaceException	`spaceId, userId`	void	Revoke a request to join a space. You should use removePendingUser(org.exoplatform.socia String) instead. It will be removed in eXo Social 1.3.x.
inviteMember(Space space, String userId) throws SpaceException	`space, userId`	void	Invite a userId to become a member of a space. You should use addInvitedUser(org.exoplatform.social.core String) instead. It will be removed in eXo Social 1.3.x.
inviteMember(String spaceId, String userId) throws SpaceException	`spaceId, userId`	void	Invite a userId to a be member of a space. You should use addInvitedUser(org.exoplatform.social.core String) instead. It will be removed in eXo Social 1.3.x.
revokeInvitation(Space space, String userId) throws SpaceException	`space, userId`	void	Revoke an invitation. Remove a user from the invited member list of the space. You should use removeInvitedUser(org.exoplatform.social.

Method	Param	Return	Description
			String) instead. It will be removed in eXo Social 1.3.x.
revokeInvitation(String spaceId, String userId) throws SpaceException	`spaceId, userId`	void	Revoke an invitation. Remove a user from the invited member list of the space. You should use removeInvitedUser(org.exoplatform.social. String) instead. It will be removed in eXo Social 1.3.x.
acceptInvitation(Space space, String userId) throws SpaceException	`space, userId`	void	Accept an invitation and move a user from the invited list to the member list. You should use addMember(org.exoplatform.social.core.sp String) instead. It will be removed in eXo Social 1.3.x.
acceptInvitation(String spaceId, String userId) throws SpaceException	`spaceId, userId`	void	Accept an invitation and move a user from the invited list to the member list. You should use addMember(org.exoplatform.social.core.sp String) instead. It will be removed in eXo Social 1.3.x.
denyInvitation(Space space, String userId) throws SpaceException	`space, userId`	void	Deny an invitation and remove a user from the invited list. You should use removeInvitedUser(org.exoplatform.social. String) instead. It will be removed in eXo Social 1.3.x.
denyInvitation(String spaceId, String	`spaceId, userId`	void	Deny an invitation and remove a user from

Method	Param	Return	Description
userId) throws SpaceException			the invited list. You should use removeInvitedUser(org.exoplatform.social. String) instead. It will be removed in eXo Social 1.3.x.
validateRequest(Space space, String userId) throws SpaceException	`space, userId`	void	Validate a request and move a user from the pending list to the member list. You should use addMember(org.exoplatform.social.core.sp String) instead. It will be removed in eXo Social 1.3.x.
validateRequest(String spaceId, String userId) throws SpaceException	`spaceId, userId`	void	Validate a request and move a user from the pending list to the member list. You should use addMember(org.exoplatform.social.core.sp String) instead. It will be removed in eXo Social 1.3.x.
declineRequest(Space space, String userId) throws SpaceException	`space, userId`	void	Decline a request and remove a user from the pending list. You should use removePendingUser(org.exoplatform.socia String) instead. It will be removed in eXo Social 1.3.x.
declineRequest(String spaceId, String userId) throws SpaceException	`spaceId, userId`	void	Decline a request and remove a user from the pending list. You should use removePendingUser(org.exoplatform.socia String) instead. It will be removed in eXo Social 1.3.x.

Method	Param	Return	Description
registerSpaceLifeCycleListener(SpaceLifeCycleListener listener)	listener	void	Register a space lifecycle listener. Deprecated: it will be removed in eXo Social 1.3.x.
unregisterSpaceLifeCycleListener(SpaceLifeCycleListener listener)	listener	void	Unregister a space lifecycle listener. Deprecated: it will be removed in eXo Social 1.3.x.
setPortletsPrefsRequired(PortletPreferenceRequiredPlugin portletPrefsRequiredPlugin)	portletPrefsRequiredPlugin	void	Set the portlet preferences got from the plug-in configuration. You should use SpaceApplicationConfigPlugin(org.exoplat String) instead. It will be removed in eXo Social 1.3.x.
getPortletsPrefsRequired()	null	String	Get the portlet preferences required to use in creating the portlet application. Deprecated: it will be removed in eXo Social 1.3.x.

Java APIs sample code/ tutorial

Activity Stream

eXo Social provides users with a way to share their activity information (also known as Activity Stream) and collaborate in spaces (also known as group work). With the API, you can customize the way to display activities or publish new ones. To manipulate activities, you need to use the *AtivityManager* service.

Publish an activity

There are two types of activities: activities for a user and activities for a space. The following examples will show you how to publish an activity for each type.

Publish an activity for a user

```
package org.exoplatform.publish.user;

import org.exoplatform.services.log.ExoLogger;
import org.exoplatform.services.log.Log;

import org.exoplatform.container.PortalContainer;
import org.exoplatform.social.core.manager.ActivityManager;
import org.exoplatform.social.core.manager.IdentityManager;
import org.exoplatform.social.core.identity.model.Identity;
import org.exoplatform.social.core.activity.model.ExoSocialActivity;
import org.exoplatform.social.core.activity.model.ExoSocialActivityImpl;
import org.exoplatform.social.core.identity.provider.OrganizationIdentityProvider;

public class PublishActivityForUser {
  // Exo Log.
  private final Log LOG = ExoLogger.getLogger(PublishActivityForUser.class);

  // Portal container.
  private PortalContainer container;

  // identityManager manages identities.
  private IdentityManager identityManager;

  // activityManager manages activities.
  private ActivityManager activityManager;

  private final static String DEFAULT_USER_NAME = "zun";
  private final static String DEFAULT_ACTIVITY_TITLE = "Hello World!";

  /**
   * Constructor.
   */
  public PublishActivityForUser() {
    // Gets the current container.
    container = PortalContainer.getInstance();

    // Gets identityManager to handle an identity operation.
   identityManager = (IdentityManager) container.getComponentInstance(IdentityManager.class);

    // Gets activityManager to handle an activity operation.
```

```
                                        activityManager = (ActivityManager)
container.getComponentInstanceOfType(ActivityManager.class);
 }

 public void createActivityForUser() {
   try {
     // Gets an existing identity or creates a new one.
                                                  Identity userIdentity =

DEFAULT_USER_NAME, false);

     // Creates a new activity for this user.
     ExoSocialActivity activity = new ExoSocialActivityImpl();
     activity.setUserId(userIdentity.getId());
     activity.setTitle(DEFAULT_ACTIVITY_TITLE);
     // Saves an activity into JCR by using ActivityManager.
     activityManager.saveActivity(activity);
   } catch (Exception e) {
     LOG.error("can not save activity.", e);
   }
 }
}
```

Publish an activity for a space

```
package org.exoplatform.publish.space;

import org.exoplatform.services.log.ExoLogger;
import org.exoplatform.services.log.Log;

import org.exoplatform.container.PortalContainer;
import org.exoplatform.social.core.manager.ActivityManager;
import org.exoplatform.social.core.manager.IdentityManager;
import org.exoplatform.social.core.identity.provider.OrganizationIdentityProvider;
import org.exoplatform.social.core.identity.model.Identity;
import org.exoplatform.social.core.activity.model.ExoSocialActivity;
import org.exoplatform.social.core.activity.model.ExoSocialActivityImpl;

import org.exoplatform.social.core.space.model.Space;
import org.exoplatform.social.core.space.SpaceException;
import org.exoplatform.social.core.space.spi.SpaceService;
```

```
import org.exoplatform.social.core.identity.provider.SpaceIdentityProvider;

public class PublishActivityForSpace {
  // Exo Log.
  private final Log LOG = ExoLogger.getLogger(PublishActivityForSpace.class);

  // Portal container.
  private PortalContainer container;

  // identityManager manages identities.
  private IdentityManager identityManager;

  // activityManager manages activities.
  private ActivityManager activityManager;

  // spaceService manages spaces.
  private SpaceService spaceService;

  private final static String DEFAULT_NAME_SPACE = "mySpace";
  private final static String DEFAULT_USER_NAME = "zun";
  private final static String DEFAULT_ACTIVITY_TITLE = "An activity for space";

  /**
   * Constructor method.
   */
  public PublishActivityForSpace() {
    // Gets the current container.
    container = PortalContainer.getInstance();

    // Gets identityManager to manage identities.
   identityManager = (IdentityManager) container.getComponentInstance(IdentityManager.class);

    // Gets activityManager to manage activities.
                                              activityManager          =          (ActivityManager)
 container.getComponentInstanceOfType(ActivityManager.class);

    // Gets spaceService to handle the operation of a space.
   spaceService = (SpaceService) container.getComponentInstanceOfType(SpaceService.class);
  }

  public void createActivityForSpace() {
    try {
      // make sure that a space with the name "mySpace" is created.
      Space space = spaceService.getSpaceByDisplayName(DEFAULT_NAME_SPACE);
```

```
      if (space != null) {
        // Gets spaceIdentity if it already exists. If not, a new one is created.
        Identity spaceIdentity = identityManager.getOrCreateIdentity(SpaceIdentityProvider.NAME,
 DEFAULT_NAME_SPACE, false);
        // Gets an identity if it already exists. If not, a new one is created.
                                                                Identity          userIdentity          =

 DEFAULT_USER_NAME, false);
        // Creates a new activity for this space.
        ExoSocialActivity activity =  new ExoSocialActivityImpl();
        activity.setUserId(userIdentity.getId());
        activity.setTitle(DEFAULT_ACTIVITY_TITLE);
        activityManager.saveActivity(spaceIdentity, activity);
      }
    } catch (SpaceException e) {
      LOG.error("Can not save activity.", e);
    } catch (Exception e) {
      LOG.error("Can not save activity.", e);
    }
  }
}
```

Configure an activity processor

An activity processor is used to modify the content of activities before they are returned from an activity manager. For example, to create an activity processor to replace all the texts representing the smile face ":-)" in the activity title by the smiley icons, do as follows:

Firstly, create the *SmileyProcessor* class by extending the *BaseActivityProcessorPlugin*.

```
package org.exoplatform.social.core.activitystream;

import org.exoplatform.social.core.BaseActivityProcessorPlugin;
import org.exoplatform.container.xml.InitParams;
import org.exoplatform.social.core.activity.model.ExoSocialActivity;

public class SmileyProcessor extends BaseActivityProcessorPlugin {
  private String smiley;

  public SmileyProcessor(InitParams params) {
    super(params);
    this.smiley = "<img src={{{"}}}http://www.tombraider4u.com/pictures/smiley.gif{{{"}}}/>";
  }
```

```
  @Override
  public void processActivity(ExoSocialActivity activity) {
    String title = activity.getTitle();
    activity.setTitle(title.replaceAll(":-)", this.smiley));
  }
}
```

Then, register this processor by editing the *configuration.xml* file:

```
<external-component-plugins>
  <target-component>org.exoplatform.social.core.manager.ActivityManager</target-component>
  <component-plugin>
    <name>SmileyProcessor</name>
    <set-method>addProcessorPlugin</set-method>
    <type>org.exoplatform.social.core.activitystream.SmileyProcessor</type>
    <description/>
    <init-params>
      <values-param>
        <name>priority</name>
        <value>1</value>
      </values-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

The "init-params" contains all the key-value data which a processor will use to initialize. In the above configuration, the **priority** value indicates the order in which this processor is executed. If the value is 1, this processor will be used before all the remaining processors with the lower priority.

Publish an RSS feed with feedmash

It is really easy to publish an RSS feed to a space's activity stream. eXo Social provides *FeedmashJobPlugin* to publish the RSS feeds. As you can see in the project "*exo.social.extras.feedmash*", there are the *JiraFeedConsumer* and *HudsonFeedConsumer* samples to post eXo Social project's feeds (jira and hudson) to a pre-defined space named *exosocial* in a specific portal container named *socialdemo* as in the configuration file:

```
<external-component-plugins>
            <target-component>org.exoplatform.services.scheduler.JobSchedulerService</target-component>
  <component-plugin>
    <name>RepubSocialJiraActivityJob</name>
    <set-method>addPeriodJob</set-method>
    <type>org.exoplatform.social.extras.feedmash.FeedmashJobPlugin</type>
    <description/>
    <init-params>
      <properties-param>
        <name>mash.info</name>
                  <property name="feedURL" value="http://jira.exoplatform.org/plugins/servlet/streams?key=SOC"/>
        <property name="categoryMatch" value="resolved|created"/>
        <property name="targetActivityStream" value="space:exosocial"/>
        <property name="portalContainer" value="socialdemo"/>
      </properties-param>
      <properties-param>
        <name>job.info</name>
        <description>save the monitor data periodically</description>
        <property name="jobName" value="JIRAFeedConsumer"/>
        <property name="groupName" value="Feedmash"/>
        <property name="job" value="org.exoplatform.social.feedmash.JiraFeedConsumer"/>
        <property name="repeatCount" value="0"/>
        <property name="period" value="60000"/>
        <property name="startTime" value="+45"/>
        <property name="endTime" value=""/>
      </properties-param>
    </init-params>
  </component-plugin>
  <component-plugin>
    <name>WatchSocialBuildStatus</name>
    <set-method>addPeriodJob</set-method>
    <type>org.exoplatform.social.extras.feedmash.FeedmashJobPlugin</type>
    <description/>
    <init-params>
      <properties-param>
        <name>mash.info</name>
          <property name="feedURL" value="http://builder.exoplatform.org/hudson/view/social/job/social-trunk-ci/rssAll"/>
        <property name="targetActivityStream" value="space:exosocial"/>
        <property name="portalContainer" value="socialdemo"/>
      </properties-param>
      <properties-param>
```

```
      <name>job.info</name>
      <description>save the monitor data periodically</description>
      <property name="jobName" value="HudsonFeedConsumer"/>
      <property name="groupName" value="Feedmash"/>
      <property name="job" value="org.exoplatform.social.feedmash.HudsonFeedConsumer"/>
      <property name="repeatCount" value="0"/>
      <property name="period" value="60000"/>
      <property name="startTime" value="+100"/>
      <property name="endTime" value=""/>
    </properties-param>
  </init-params>
 </component-plugin>
</external-component-plugins>
```

Run eXo Social with the URL: *http://localhost:8080/socialdemo*, then log in and create a space named "exosocial". After creating the "exosocial" space, all the feeds of the eXo Social project on Jira and Hudson will be automatically published to the *exosocial* space.

Sample Code

See the following code snippet to know more details about how to publish an activity and add comments to an activity:

```
package org.exoplatform.social.introduction.activitystreamandexosocialactivity;

import org.exoplatform.services.log.ExoLogger;
import org.exoplatform.services.log.Log;

import org.exoplatform.container.PortalContainer;
import org.exoplatform.social.core.activity.model.ActivityStream;
import org.exoplatform.social.core.activity.model.ExoSocialActivity;
import org.exoplatform.social.core.activity.model.ExoSocialActivityImpl;
import org.exoplatform.social.core.manager.ActivityManager;
import org.exoplatform.social.core.manager.IdentityManager;
import org.exoplatform.social.core.identity.model.Identity;
import org.exoplatform.social.core.identity.provider.OrganizationIdentityProvider;

public class IntroduceActivityStreamAndExoSocialActivity {
 // Exo Log.
                        private final Log LOG =
 ExoLogger.getLogger(IntroduceActivityStreamAndExoSocialActivity.class);

 // Demo identity.
```

```
 private Identity demoIdentity;

 // John identity.
 private Identity johnIdentity;

 // identityManager manages identities.
 private IdentityManager identityManager;

 // activityManager manages activities.
 private ActivityManager activityManager;

 // Portal container.
 private PortalContainer container;

 private final static String DEMO_NAME = "demo";
 private final static String JOHN_NAME = "john";
 private final static String DEFAULT_ACTIVITY_TITLE = "blabla";
 private final static String DEFAULT_COMMENT_TITLE = "comment blah blah";

 /**
  * Constructor.
  */
 public IntroduceActivityStreamAndExoSocialActivity() {
   // Gets the current container.
   container = PortalContainer.getInstance();

   // Gets IdentityManager to handle an identity operation.
                                              identityManager              =              (IdentityManager)
container.getComponentInstanceOfType(IdentityManager.class);

   // Gets ActivityManager to handle activity operation.
                                  ActivityManager          activityManager          =          (ActivityManager)
container.getComponentInstanceOfType(ActivityManager.class);

  // Gets or create demo's identity
      demoIdentity   =   identityManager.getOrCreateIdentity(OrganizationIdentityProvider.NAME,
DEMO_NAME, false);

   // Gets or creates the identity "john".
        johnIdentity   =   identityManager.getOrCreateIdentity(OrganizationIdentityProvider.NAME,
JOHN_NAME, false);
 }

  /**
```

```
  * Posts activity to activity stream
  */
  public void introduceActivityStreamAndExoSocialActivity {

     // Sets a string that specifies the primary text of an activity. This field is REQUIRED by ActivityManager. The title field may only have the following HTML tags: <b> <i>, <a>, <span>.
    activity.setTitle(DEFAULT_ACTIVITY_TITLE);

    // Sets this activity for demo.
   activity.setUserId(demoIdentity.getId());

    // Saves the activity.
    activityManager.saveActivity(johnIdentity, activity);

    // Gets activity stream.
    ActivityStream activityStream = activity.getActivityStream();

    // Type of the activity stream. It can be organization or space.
    LOG.info("activity stream type: " + activityStream.getType());

    LOG.info("activity stream id: " + activityStream.getId());
    LOG.info("activity stream pretty id: " + activityStream.getPrettyId());
    LOG.info("activity stream perma link: " + activityStream.getPermaLink());

    LOG.info("activity stream id: " + activity.getStreamId());
    LOG.info("activity stream owner: " + activity.getStreamOwner());

    // Comment in Social
    ExoSocialActivity demoActivity = new ExoSocialActivityImpl();;
    activity.setTitle(DEFAULT_ACTIVITY_TITLE);
    activityManager.saveActivity(demoIdentity, demoActivity);

    ExoSocialActivity comment = new ExoSocialActivityImpl();;
    comment.setTitle(DEFAULT_COMMENT_TITLE);

    //Sets comment of demo
   comment.setUserId(demoIdentity.getId());

    //Saves a comment.
    activityManager.saveComment(activity, comment);
  }
}
```

OpenSocial

eXo Social supports the OpenSocial standard. So you can integrate OpenSocial gadgets in your dashboard and use the RPC or REST service to view or publish the social data. With the support for the OpenSocial standard, eXo Social provides a framework for developers to build gadgets that can display and mash up activity information for contacts, social networks, applications and services.

Gadget

Gadgets are web-based software components based on HTML, CSS, JavaScript; defined by using an XML declaration syntax. They allow developers to easily write social applications that work on the social networks supporting OpenSocial APIs without modification. See the following links for detailed information:

- *Gadgets Specification* [http//code:google.com-apis-gadgets-docs-spec.html]
- *OpenSocial Core Gadget Specification v0.8* [http://www.opensocial.org/Technical-Resources/opensocial-spec-v08/gadget-spec.html]
- *Gadgets APIs* [http://www.opensocial.org/Technical-Resources/opensocial-spec-v08/gadgets-reference08]

To know how to create an OpenSocial gadget, see *here* [http://wiki.opensocial.org/index.php?title=Gadget_Developer's_Guide].

Note

Gadgets will work out of the box on Dashboard. In eXo, gadgets are wrapped by GadgetWrapperPortlet so they can work as any other portlet applications. At present, eXo Social supports *OpenSocial v0.8.* [http://www.opensocial.org/Technical-Resources/opensocial-spec-v08.html]

Supported APIs

eXo Social leverages *Apache Shindig* [http://shindig.apache.org/overview.html] - an OpenSocial reference implementation to provide and extend OpenSocial APIs which is compatible with the common OpenSocial APIs which is supported by other big social networks like *Ning* [http://www.ning.com], *Hi5* [http://www.hi5.com], *Orkut* [http://www.orkut.com] and more.

To get more details about Supported APIs, refer to *Opensocial Specs* [http://www.opensocial.org/page/specs-1].

REST/RPC API

Suppose that you are running the local host at port 8080 (*http://localhost:8080/)*, the path of the API will be:

- REST API:

http://localhost:8080/social/social/rest

- RPC API:

http://localhost:8080/social/social/rpc

To learn what you can do with the APIs, have a look at the *specification* [http://opensocial-resources.googlecode.com/svn/spec/1.0/Social-Data.xml]. If you are developing in Java, you can use the *opensocial-java-client* [http://code.google.com/p/opensocial-java-client/].

Configure the security

If you are using OpenSocial, you need to configure the OAuth authentication. With the case of eXo Platform, you need to edit the file: *gatein/conf/portal/portal/configuration.xml* and add the following configuration:

```
<component>
  <key>org.exoplatform.social.opensocial.oauth.ServiceProviderStore</key>
  <type>org.exoplatform.social.opensocial.oauth.ServiceProviderStore</type>
  <init-params>
    <properties-param>
      <name>grails-book-flow</name>
      <description>consumer key and secret for sample oauth provider. </description>
      <property name="consumerKey" value="YOUR_KEY_HERE"/>
      <property name="sharedSecret" value="YOUR_SECRET_KEY_HERE"/>
    </properties-param>
  </init-params>
</component>
```

consumerKey and *sharedSecret* are keys that need to be shared with the application which is doing the request.

Publish an activity into a space

This functionality is not available in the standard OpenSocial APIs.

Instead of publishing your activities to the group @self as usual, you will publish them to the group "space:spaceId" or "space:spacePrettyName".

After using the OpenSocial Java library and Groovy, your code will look like this:

```
```

```
    def client = getOpenSocialClient()

    //create your new activity
    Activity activity = new Activity()
  activity.body = "xx purchased the book xxx"
    activity.title = "BookFlow Purchased"

    //prepare the request that will create the activity
    Request request = ActivitiesService.createActivity(activity);
    //specify that the creation of this new activity is for the space bookflow
    request.groupId = "space:bookflow";

    client.send(request);
```

In the example above, the groupId is set to "space:bookflow" and bookflow is the name of the space.

Tutorial

- See *Grails + eXo Social tutorial.* [http://www.exoplatform.com/company/public/website/resource-viewer?path=/website/Content%20types/Tutorial/grails-exo-social]

People

eXo Social provides a way to manage profile information, and connections between users. With the APIs provided, you can easily add, manage and customize information and relationships of users.

Identity

The identity allows identifying a unique social object. The eXo Social objects can be person, group, application, or whatever related to social interactions, such as connecting, publishing an activity stream or holding a user profile.

IdentityProvider

eXo Social provides IdentityProvider as an identiy mechanism for the third parties to extend eXo Social's identity system without any limitation. Here is an example:

```
class SampleIdentityProvider extends OrganizationIdentityProvider{
  public SampleIdentityProvider(OrganizationService organizationService) {
    super(organizationService);
```

```
  }

  @Override
  public void populateProfile(Profile profile, User user) {
    profile.setProperty(Profile.FIRST_NAME, "this is first name");
    profile.setProperty(Profile.LAST_NAME, "this is last name");
    profile.setProperty(Profile.USERNAME, "this is user name");
    profile.setProperty(Profile.URL, "/path/to/profile/");
  }
}
```

In this example, the *SampleIdentityProvider* class extends *OrganizationIdentityProvider* which is the IdentityProvider used to connect to the portal user's base. In this class, the *populateProfile* method is overridden and some dummy data are added to the profile fields.

IdentityManager

IdentityManager is the service used to manipulate the identity operations, such as creating, getting, deleting or finding a profile. You can get the IdentityManager via the ExoContainer. The following code will show how to get an IdentityManager instance and create a basic identity instance:

```
import org.exoplatform.container.ExoContainer;
import org.exoplatform.container.ExoContainerContext;
import org.exoplatform.social.core.identity.IdentityManager;
import org.exoplatform.social.core.identity.impl.organization.OrganizationIdentityProvider;
import org.exoplatform.social.core.identity.model.Identity;

//.....

String containerName = "portal";
String username = "zun";

//get container to get other registered components
ExoContainer container = ExoContainerContext.getContainerByName(containerName);

//get IdentityManager to handle identity operation
IdentityManager identityManager = (IdentityManager)
 container.getComponentInstanceOfType(IdentityManager.class);

//get ActivityManager to handle activity operation
ActivityManager activityManager = (ActivityManager)
 container.getComponentInstanceOfType(ActivityManager.class);
```

```
//create new user with name Zun
Identity userIdentity = identityManager.getOrCreateIdentity(OrganizationIdentityProvider.NAME,
 username);
```

ProfileListener

APIs provide notification interfaces which you can implement to create your own handlers for notification when having the profile modifications by extending the *ProfileListenerPlugin* class, or relationship changes by extending *RelationshipListenerPlugin.*

1. Create the *ProfileLoggerListener* class to log all profile modifications of the systems. The abstract class named *ProfileListenerPlugin* provides the interface to implement this method.

```
import org.exoplatform.services.log.ExoLogger;
import org.exoplatform.services.log.Log;
import org.exoplatform.social.core.identity.lifecycle.ProfileListenerPlugin;
import org.exoplatform.social.core.identity.spi.ProfileLifeCycleEvent;


public class ProfileLoggerListener extends ProfileListenerPlugin{
  private static final Log logger = ExoLogger.getExoLogger(ProfileLoggerListener.class);
  @Override
  public void avatarUpdated(ProfileLifeCycleEvent event) {
    logger.info("@" + event.getUsername() + " profile has updated his basic profile info.");
  }

  @Override
  public void basicInfoUpdated(ProfileLifeCycleEvent event) {
    logger.info("@" + event.getUsername() + " profile has updated his basic profile info.");
  }

  @Override
  public void contactSectionUpdated(ProfileLifeCycleEvent event) {
    logger.info("@" + event.getUsername() + " profile has updated his contact info.");
  }

  @Override
  public void experienceSectionUpdated(ProfileLifeCycleEvent event) {
    logger.info("@" + event.getUsername() + " profile has an updated experience section.");
  }

  @Override
```

```
  public void headerSectionUpdated(ProfileLifeCycleEvent event) {
    logger.info("@" + event.getUsername() + " has updated his header info.");
  }
}
```

2. Add some configurations to this class to the configuration.xml:

```
<external-component-plugins>
  <target-component>org.exoplatform.social.core.identity.IdentityManager</target-component>
  <component-plugin>
    <name>ProfileLoggerListener</name>
    <set-method>addProfileListener</set-method>
    <type>path.to.ProfileLoggerListener</type>
  </component-plugin>
</external-component-plugins>
```

Connections

Similarly, you can apply the above steps to implement *RelationshipListenerPlugin* for relationship notifications.

Relationship is the bridge between two identities in eXo Social. There are many types of relationships defined in the Relationship class. With these types, a user can invite another user, confirm invitations or remove relationship.

Users connection

The following code will show you how to invite a user to connect with another:

```
import org.exoplatform.container.ExoContainer;
import org.exoplatform.container.ExoContainerContext;
import org.exoplatform.social.core.identity.IdentityManager;
import org.exoplatform.social.core.identity.impl.organization.OrganizationIdentityProvider;
import org.exoplatform.social.core.identity.model.Identity;
import org.exoplatform.social.core.relationship.Relationship;
import org.exoplatform.social.core.relationship.RelationshipManager;

public void inviteUser() throws Exception {
  String containerName = "portal";
  ExoContainer container = ExoContainerContext.getContainerByName(containerName);
```

```
                IdentityManager identityManager = (IdentityManager)
container.getComponentInstanceOfType(IdentityManager.class);

                                Identity invitedIdentity =
identityManager.getOrCreateIdentity(OrganizationIdentityProvider.NAME, "Hoat");
 String invitedUserId = invitedIdentity.getId();

 String currUserId = "Zun";
                                Identity currentIdentity =
identityManager.getOrCreateIdentity(OrganizationIdentityProvider.NAME,  currUserId);

            RelationshipManager relationshipManager = (RelationshipManager)
container.getComponentInstanceOfType(RelationshipManager.class);
 Relationship relationship = relationshipManager.invite( currentIdentity, invitedIdentity);
}
```

RelationshipListener

The following example will show you how to implement one of these interfaces and how to configure this plugin. The following step is similar to the Profile notifications implementation.

1. Create the *RelationshipLoggerListener* class:

```
import org.exoplatform.social.core.relationship.Relationship;
import org.exoplatform.social.core.relationship.lifecycle.RelationshipListenerPlugin;
import org.exoplatform.social.relationship.spi.RelationshipEvent;

public class RelationshipLoggerListener extends RelationshipListenerPlugin{
 private static final Log logger = ExoLogger.getExoLogger(RelationshipLoggerListener.class);

 @Override
 public void confirmed(RelationshipEvent event) {
  String[] names = getUserNamesFromEvent(event);
  logger.info(names[0] +" confirmed the invitation of "+ names[1]);
 }

 @Override
 public void ignored(RelationshipEvent event) {
  String[] names = getUserNamesFromEvent(event);
  logger.info(names[0] +" ignored the invitation of "+ names[1]);
 }

 @Override
```

```
  public void removed(RelationshipEvent event) {
    String[] names = getUserNamesFromEvent(event);
    logger.info(names[0] +" removed the relationship with "+ names[1]);
  }

  private String[] getUserNamesFromEvent(RelationshipEvent event){
    Relationship relationship = event.getPayload();

    Identity id1 = relationship.getIdentity1();
    Identity id2 = relationship.getIdentity2();

    String user1 = "@" + id1.getRemoteId();
    String user2 = "@" + id2.getRemoteId();

    return new String[]{user1, user2 };
  }
}
```

2. Add some configurations for this class in the *configuration.xml* file:

```
<external-component-plugins>
        <target-component>org.exoplatform.social.core.relationship.RelationshipManager</target-component>
  <component-plugin>
    <name>RelationshipLoggerListener</name>
    <set-method>addListenerPlugin</set-method>
    <type>classpath.of.your.RelationshipLoggerListener</type>
  </component-plugin>
</external-component-plugins>
```

Spaces

eXo Social provides a way to create groups and to share data and applications by spaces. A space has its own activity stream in which applications or members can publish information. In each space, members use applications together with shared data.

To manipulate the spaces, you will use the SpaceService. To get an instance of this class, you need to get the current *PortalContainer* instance.

Spaces Management

The following example will show how to create a space:

```
package org.exoplatform.social.sample;

import org.exoplatform.container.PortalContainer;
import org.exoplatform.social.application.impl.DefaultSpaceApplicationHandler;
import org.exoplatform.social.space.Space;
import org.exoplatform.social.space.SpaceException;
import org.exoplatform.social.space.SpaceService;

public class SpaceCreationSample {

  public void createSpace() throws SpaceException {
    String spaceName = "mySpace";
    String creator = "jeremi";
    PortalContainer container = PortalContainer.getInstance();
                                    SpaceService spaceService = (SpaceService)
 container.getComponentInstanceOfType(SpaceService.class);
    // verify if there is no space already created
    Space space = spaceService.getSpaceByDisplayName(spaceName);
    if (space == null) {
      space = new Space();
      space.setDisplayName(spaceName);
      space.setRegistration(Space.OPEN);
      space.setDescription("space description");
       //DefaultSpaceApplicationHander is the default implementation of SpaceApplicationHandler.
  You can create your own by extending SpaceApplicationHandler. The default type is "classic"
 (DefaultSpaceApplicationHandler.NAME = clasic)
      space.setType(DefaultSpaceApplicationHandler.NAME);
      //create the space
      space = spaceService.createSpace(space, creator);
      //initialize the applications
      spaceService.initApps(space);

    }
  }
}
```

Space's applications management

Add an application to a space

You can add portlet or gadget applications to spaces. Once added, all members of the space can use that application. The following code shows you how to add an application to a space:

```
public void addApplicationToSpace() throws SpaceException {
  //Your portlet name
  String appId = "SocialBannerPortlet";
  String spaceId = "zunSpace";

  //get container to get other registered components
  PortalContainer container = PortalContainer.getInstance();

  //get space service for installing operations
                    SpaceService spaceService = (SpaceService)
 container.getComponentInstanceOfType(SpaceService.class);

  //install application for the space
  spaceService.installApplication(spaceId, appId);

  //you must activate installed application to be able to use it
  spaceService.activateApplication(spaceId, appId);
}
```

Note

appId is the portlet or gadget name as defined in *portlet.xml* or *gadget.xml* in the web-app. You can find it in *social-portlet.war* and *social.war*.

Remove an application from a space

You can remove portlet or gadget applications from a space and the members of this space will not see them anymore. The following code shows you how to remove an application from a space:

```
public void removeApplicationFromSpace() throws SpaceException {
  //Your portlet name
  String appId = "SocialBannerPortlet";
  String appName = "SocialBannerPortlet1"
  String spaceId = "zunSpace";

  //get container to get other registered components
  PortalContainer container = PortalContainer.getInstance();

  //get space service for installing operations
                    SpaceService spaceService = (SpaceService)
 container.getComponentInstanceOfType(SpaceService.class);
```

```
  //install application for the space
  spaceService.removeApplication(spaceId, appId, appName);
}
```

Space's members management

SpaceService allows you to manage the spaces' members. Here is the way to add a new member to a space:

```
String spacePrettyName = "mySpace";

PortalContainer container = PortalContainer.getInstance();
SpaceService spaceService = (SpaceService)
 container.getComponentInstanceOfType(SpaceService.class);

Space space = service.getSpaceByPrettyName(spacePrettyName);
if (space != null) {
    spaceService.addMember(space, "mary");
}
```

Listener to a space lifecycle

To receive notifications of what are happening in spaces, you need to extend *SpaceListenerPlugin* and register it. Every method takes a *SpaceLifeCycleEvent* object as a parameter that contains the information about the event and its context.

The available events are:

- *spaceCreated*: This event is called right after a space is created successfully with its applications.
- *spaceRemoved*: This event is called right after the space is removed. It means all the applications of the space are removed, its group and group navigation are removed.
- *applicationAdded*: This event is called right after an application is added (installed) to the space.
- *applicationActivated*: This event is called right after an application is activated in the space.
- *applicationDeactivated*: This event is called right after an application is deactivated.
- *applicationRemoved*: This event is called right after an application is removed from the space.
- *joined*: This event is called right after a user joins the space.
- *left*: This event is called right after a space member leaves its space.

- *grantedLead*: This event is called right after a user is granted the space's manager role.
- *revokedLead*: This event is called right after the space's manager is revoked his role to be a space member.

```
import org.exoplatform.social.space.lifecycle.SpaceListenerPlugin;

public class MySpaceListenerPlugin extends SpaceListenerPlugin {
...
}
```

As an example, see *SpaceActivityPublisher* [http://fisheye.exoplatform.org/browse/social/trunk/component/exosocial/src/main/java/org/exoplatform/social/space/SpaceActivityPublisher.java?r=HEAD] that publishes an activity based on an event that happened in a space.

To register your listener, configure it as a plugin to the SpaceService like this:

```
<external-component-plugins>
  <target-component>org.exoplatform.social.core.space.spi.SpaceService</target-component>
  <component-plugin>
    <name>SpaceActivityPublisher</name>
    <set-method>addSpaceListener</set-method>
    <type>org.mycompany.MySpaceListenerPlugin</type>
  </component-plugin>
</external-component-plugins>
```

Space widget tutorial

The eXo Social widget enables developers to add capabilities of eXo Social to external applications. Since the widget is hosted on your eXo Social server, it can display personalized information. An activity stream of the most recent user actions will display to the group's members.

There are two options of the eXo Social widget that provide different levels of integration and information.

The basic version of this widget is an iFrame. The more advanced version is a button you can insert in a page; this will display a small pop-up with information about the space.

Basic version

To insert the basic version, you need to have an iFrame to insert on your site.

```
<iframe scrolling="no" height="180" frameborder="no" width="220" src="http://URL_OF_YOUR_EXO_INSTALLATION.COM/rest/private/spaces/portal/space_info?spaceName=NAME_OF_YOUR_SPACE&description=DESCRIPTION_OF_THE_SPACE"></iframe>
```

To install this version in your application, replace all the uppercase text below:

- *URL_OF_YOUR_EXO_INSTALLATION.COM*: This is the URL of your eXo Social installation. If you are testing on your local computer, the URL may be *localhost:8080*.
- *NAME_OF_YOUR_SPACE*: This is the title of your space in eXo Social. In the URL, it is necessary to avoid special characters. For the space name, you can only use alphanumeric characters and "_", ".", "-" or ".".
- *DESCRIPTION_OF_THE_SPACE*: This will be displayed in the list of spaces.

Advanced version

To install an advanced version of the widget, you need to insert a code snippet in your page. This includes some HTMLs plus some JavaScript. The necessary CSS will be added dynamically.

Next, insert the following code at the position you want the button to be displayed:

```
<div class="exoSpacesContainer"><a href="javascript:void(0);" id="exoSpacesLink" class="exoSpacesLink" target="_blank">Space</a></div>
<script src="/socialWidgetResources/javascript/space.js"></script>
<script>spaces.createPopup("exoSpacesLink", "MyAppName - my social object", "my cool description");</script>
```

The important function here is:

```
spaces.createPopup(link, spaceName, description)
```

In which:

- *link* is the ID or the HTMLElement where the pop-up will be placed. If you copy and paste the code snippet provided above, you do not need to change this value.
- *spaceName* is the name of space. It is also used to identify the space. You should use the following format: "MyAppName - my social object".

- *description* is a brief introduction about your space.

Configure

- serverURL (Default: *"http://127.0.0.1:8080"*): The address of your eXo installation. To change it, use *spaces.setServerURL(...)*.
- spaceServicePath (Default: *"/rest/private/spaces/"*): The path to the spaces service. It is rare you have to change it; but if needed, use *spaces.setSpaceServicePath(...).*
- portalName (Default: *"socialdemo"*): The name of portal you are using. To change it, use *spaces.setPortalName(...)*.

If you want to change any part of this configuration, the best way is to change before creating the pop-up. For example:

```
<div class="exoSpacesContainer"><a href="#" id="exoSpacesLink" class="exoSpacesLink"
 target="_blank">Space</a></div>
<script src="/socialWidgetResources/javascript/space.js"></script>
<script>
  spaces.setServerURL("http://192.168.2.100:8080");
  spaces.createPopup("exoSpacesLink", "My cool new space", "my cool description");
</script>
```

You can see an example of integration at: *http://localhost:8080/socialWidgetResources/test.html*

How to extend the activities rendering

Objective

In this section, you will learn how to implement a preprocessor for activities rendering.

Requirements

To understand this section, the knowledge of Java and eXo Kernel (component model and its XML configuration) is prerequisite.

Why would you need to do this?

A simple activity is made of simple text. An extension point has been created at the level of activities rendering for two cases:

- support more HTML tags.
- support @mentions.

But you may want to support a special syntax, for example:

- #hashtags to feel like Twitter.
- smileys to look like Skype.
- *Markdown* [http://en.wikipedia.org/wiki/Markdown] to experience Buzz.

You can have more sophisticated cases to process, such as parsing the link that include in the activity's content. Because a process actually has the full access to the Activity, you can very well process based on the owner, app, and media item.

Write an ActivityProcessor

eXo Social lets you pre-process an activity before it is returned by the ActivityManager. To do this, you simply need to implement the interface ActivityProcessor:

```
/**
 * An activity processor is responsible to pre-process an activity before it is returned by the {@link
 ActivityManager}
 */
public interface ActivityProcessor {

  /**
   * Process an activity
   * @param activity the activity. It can be modified
   */
  void processActivity(Activity activity);

  /**
   * Priority of this processor.
   * All activity processors will be executed in ascending priority order
   * @return
   */
  int getPriority();

}
```

For example, the following shows you how to implement a *SmileyProcessor* that will replace text smileys by icons:

```
public class SmileyProcessor implements ActivityProcessor {
```

```
  String smiley =  "<img src='/images/smiley.gif'/>";

  public void processActivity(Activity activity) {
    String title = activity.getTitle();
     activity.setTitle(title.replaceAll(":-)", smiley));
  }

  public int getPriority() {
    return 100;
  }

}
```

Configure the processor

Now, you have a nice procesor, so need to hook it to the system. At runtime, the processors can be attached to *ActivityManager* via the *addProcessor(ActivityProcessor)* method.

But there is also a component plugin hooked for it: *public void addProcessorPlugin(BaseActivityProcessorPlugin plugin)*.

So, to make your processor easy to hook, you simply need to let him extend the BaseActivityProcessorPlugin.

```
public class SmileyProcessor extends BaseActivityProcessorPlugin {

  String smiley =  "<img src='/images/smiley.gif'/>";

  public SmileyProcessor(InitParams params) {
    super(params);
  }

  public void processActivity(Activity activity) {
    String title = activity.getTitle();
      activity.setTitle(title.replaceAll(":-)", smiley));
  }
}
```

It will have the additional benefit to make the priority field configurable, so you do not need to implement *getPriorty()*.

Then your processor can be configured as a component plugin like this:

```
<external-component-plugins>
            <target-component>org.exoplatform.social.core.activitystream.ActivityManager</target-component>
  <component-plugin>
    <name>SmileyProcessor</name>
    <set-method>addProcessorPlugin</set-method>
    <type>org.example.SmileyProcessor</type>
    <init-params>
      <value-param>
        <name>priority</name>
        <description>priority of this processor (lower are executed first)</description>
        <value>2</value>
      </value-param>
    </init-params>
  </component-plugin>
</external-component-plugins>
```

Restart, then place the smiley images on the server and you should see something like that:

Public REST APIs

Activities REST service

Name	Service URL	Location	Description
ActivitiesRestService	{restContextName}/{portalName}/social/activities	**Maven groupId**: org.exoplatform.social **ArtifactId**: exo.social.component.service	Provide REST services for activity applications: like/unlike; comment; delete activity.

- **API**:

Name	Service URL Endpoint	Parameters	Expected Values	Description
destroyActivity	{restContextName}/{portalName}/social/activities/destroy/{activityId}.{format}	portalName activityId format	String String String: json or xml	Destroy activity and get the JSON/XML format.

Name	Service URL Endpoint	Parameters	Expected Values	Description
showLikes	{restContextName}/{portalName}/social/activities/{activityId}/likes/show.{format}	portalName activityId format	String String String: json or xml	Show the list of likes by activityId and return the JSON/XML format.
updateLike	{restContextName}/{portalName}/social/activities/{activityId}/likes/update.{format}	portalName activityId format	String String String: json or xml	Update the list of likes by the JSON/XML format.
destroyLike	{restContextName}/{portalName}/social/activities/{activityId}/likes/destroy/{identity}.{format}	portalName activityId identityId format	String String String String: json or xml	Destroy **like** by identityId and get the JSON/XML format return format.
showComments	{restContextName}/{portalName}/social/activities/{activityId}/likes/show.{format}	portalName activityId format	String String String: json or xml	Show the comment list by the JSON/XML format.
updateComment	{restContextName}/{portalName}/social/activities/{activityId}/likes/update.{format}	portalName activityId format	String String String: json or xml	Update the comment by the JSON/XML format.
destroyComment	{restContextName}/{portalName}/social/activities/{activityId}/comments/	portalName activityId	String String	Destroy comments and return the JSON/XML format.

Name	Service URL Endpoint	Parameters	Expected Values	Description
	destroy/{commentId}.{format}	commentId format	String String: json or xml	

Example:

http://localhost:8080/rest-socialdemo/socialdemo/social/activities/s08d397dg6/likes/destroy/abc.json

Apps REST service

Name	Service URL	Location	Description
AppsRestService	{restContextName}/social/apps/	**Maven groupId**: org.exoplatform.social **ArtifactId**: exo.social.component.service	Provide REST services for the application registry gadget: shows application list.

- **API**:

Name	Service URL Endpoint	Parameters	Expected Values	Description
showApps	{restContextName}/social/apps/show.{format}	format	String: json or xml	Show applications by the JSON/XML format.

Example:

http://localhost:8080/rest-socialdemo/social/apps/show.json

Identity REST service

Name	Serive URL	Location	Description
IdentityRestService	restContextName}/{portalName}/social/identity/{username}/id	**Maven groupId**: org.exoplatform.social **ArtifactId**: exo.social.component.service	Get identityId by the username.

- **API**:

Name	Service URL Endpoint	Parameters	Expected Values	Description
UserId getId	{restContextName}/{portalName}/social/identity/{username}/id/show.json	username portalName	String String	Get the identity by username and return by the JSON format.

Example:

http://localhost:8080/rest-socialdemo/socialdemo/social/identity/john/id/show.json

Linkshare REST service

Name	Service URL	Location	Description
LinkshareRestService	{restContextName}/social/linkshare	**Maven groupId**: org.exoplatform.social **ArtifactId**: exo.social.component.service	Get information from a provided link.

- **API**:

Name	Service URL Endpoint	Parameters	Expected Values	Description
getLink	{restContextName}/social/linkshare/show.{format}	format	String: json or xml	Get the link content by posting a linkShare request.

Example:

http://localhost:8080/rest-socialdemo/social/linkshare/show.json

People Rest Service

Name	Service URL	Location	Description
PeopleRestService	{restContextName}/social/people	**Maven groupId**: org.exoplatform.social	Provide REST services for manipulating jobs related to people.

Name	Service URL	Location	Description
		ArtifactId: exo.social.component.service	

- **API**:

Name	Service URL Endpoint	Parameters	Expected Values	Description
suggestUsernames	{restContextName}/social/people/suggest.{format}	nameToSearch currentUser typeOfRelation spaceURL format	String String String String String: json or xml	Get and return a list of usernames which match the input string for suggest.

Example: *http://localhost:8080/rest-socialdemo/social/people/suggest.json*

Spaces REST service

Name	Service URL	Location	Description
SpacesRestService	{restContextName}/{portalName}/social/spaces	**Maven groupId**: org.exoplatform.social **ArtifactId**: exo.social.component.service	Provide REST services for space gadget to display users' spaces and pending spaces.

- **API**:

Name	Service URL Endpoint	Parameters	Expected Values	Description
showMySpaceList	{restContextName}/social/spaces/mySpaces/show.{format}	portalName format	String String: json or xml	Show mySpaceList by the JSON/XML format.
showPendingSpaceList	{restContextName}/social/spaces/	portalName	String	Show pendingSpaceList

Name	Service URL Endpoint	Parameters	Expected Values	Description
	pendingSpaces/show.{format}	format	String: json or xml	by the JSON/XML format.
suggestSpacenames	{restContextName}/social/spaces/spaceNames/suggest.{format}	portalName conditionToSearch typeOfRelation currentUser format	String String String String String: json or xml	Get and return space's names that match the input string for suggest.

Example:

http://localhost:8080/rest-socialdemo/social/spaces/mySpaces/show.xml

Widget Rest Service

Name	Service URL	Location	Description
WidgetRestService	{restContextName}/spaces/{containerName}	**Maven groupId**: org.exoplatform.social **ArtifactId**: exo.social.extras.widget.rest	Provide REST services for creating spaces or getting spaces'information.

- **API**:

Name	Service URL Endpoint	Parameters	Expected Values	Description
spaceInfo	{restContextName}/spaces/{containerName}/space_info	containerName portalName spacePrettyName description	String String (default value: classic) String String	Return the HTML page for displaying the information of the space. Two query parameters needed: *spaceName* and *description*.

Example:

http://localhost:8080/rest-socialdemo/spaces/socialdemo/space_info?name=Social&description=Social

Location

- **Maven groupId**: org.exoplatform.social
- **ArtifactId**: exo.social.component.service

Rest Service APIs (v1-alpha1)

Objectives:

The third parties want to integrate and extend the eXo Social capability and features and eXo Social REST Services APIs are dedicated for them to complete that. By using these REST APIs, the third parties can write any applications (desktop apps, mobile apps, web apps) to integrate with eXo Social services and business.

Conventions:

The entry point for Social Rest service must be: */{rest_context_name}/private/api/social/{version}/{portalContainerName}/{social_resources}/*. An example of activities resources: /rest/private/api/social/v1-alpha1/portal/activity/.

There are 3 types of parameters on each URL:

- **{server_params}**: this is the unchanged parameter for a specified server.
- **:id**: this param is for a specified data object, for example, activity Id and identity Id.
- **format**: this is the supported data format. It can be JSON, XML, RSS, or ATOM.

Notes:

Currently, only the basic authentication is supported. Please see more details at *http://en.wikipedia.org/wiki/Basic_access_authentication.*

The JSON support is mandatory. The other format SHOULD be supported too (XML, RSS, ATOM). The supported formats will be specified by the detailed documentation for the REST resources.

The *rest_context_name* and *portal_container_name* parameters are used as follows:

- On eXo Social standalone:

- *rest_context_name*: *rest-socialdemo*;
- *portal_container_name*: *socialdemo*;

- On eXo Platform:
  - *rest_context_name*: *rest*;
  - *portal_container_name*: *portal*;

Activity Resources

activity

POST activity.format

Description: Creates an activity to an identity's activity stream. If no *identity_id* is specified, the activity will be created to the authenticated identity's activity stream.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity.format
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity.json*
- *http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity.json*

Body Data: The data for an activity

For example:

```
{
"title": "Hello World", //required
"type": "exosocial:core", //optional
"priority": 0.5, //optional
"titleId": "", //optional
"templateParams": {
} //optional hashmap
}
```

Supported Format: JSON

Requires Authentication: true

Parameters:

- **Optional**
  - **identity_id**: The identity who posts activities to his stream, to the stream of his connection or the space stream where he is a manager or a member.

Examples

- **JSON**

Request:

```
POST: http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity.json
BODY: {"title": "Hello World!!!"}
```

Response: A list of activity objects in the JSON format.

```
{
"id": "1a2b3c4d5e6f7g8h9j",
"title": "Hello World!!!",
"appId": "",
"type": "exosocial:core",
"postedTime": 123456789, //timestamp
"createdAt": "Fri Jun 17 06:42:26 +0000 2011",
"priority": 0.5, //between 0.0 and 1.0, higher value => higher priority.
"templateParams": {},
"titleId": "",
"identityId": "123456789abcdefghi" //the identity id of the user who created this activity
}
```

GET activity/:activityId.format

Description: Gets an activity object from a specified activity Id.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity/:id.format
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e6f7g8h9i.json*
- *http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity/1a2b3c4d5e6f7g8h9i.json*

Supported Format: JSON

Requires Authentication: true

Parameters:

- **Required**
  - **id**: the Id of the existing activity.
- **Optional**
  - **poster_identity**: When this parameter is set to **true**, **t** or **1**, the returned activity will provide more information for the user who posted this activity.
  - **number_of_comments**: Specifies the number of comments to be displayed along with this activity. By default, *number_of_comments=0*. If *number_of_comments* is a positive number, this number is considered as a limit number that must be equal or less than 100. If the actual number of comments is less than the provided positive number, the number of actual comments must be returned. If the total number of comments is more than 100, it is recommended to use *activity/:id/comments.format* instead.
  - **activity_stream**: When this parameter is set to **true**, **t** or **1**, the returned activity will provide more information for the activity stream that this activity belongs to.

Examples:

- **JSON**

Request:

```
GET:  http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e6f7g8h9i.json
```

Response:

```
{
"id": "1a2b3c4d5e6f7g8h9j",
"title": "Hello World!!!",
"appId": "",
"type": "exosocial:core",
"postedTime": 123456789, //timestamp
"createdAt": "Fri Jun 17 06:42:26 +0000 2011", //The Date follows ISO 8601
"priority": 0.5, //between 0.0 and 1.0, higher value => higher priority.
"templateParams": {},
"titleId": "",
"identityId": "123456789abcdefghi", //the identity id of the user who created this activity
"liked": true, //is liked (favorites) by this authenticated identity
"likedByIdentities": ["identityId1", "identityId2"],
"posterIdentity": {}, //optional
"comments": [{}, {}, {}], //optional
"totalNumberOfComments": 1234,
"activityStream": {
"type": "user", // or "space"
"prettyId": "root", // or space_abcde
"faviconURL": "http://demo3.exoplatform.org/favicons/exo-default.jpg",
"title": "Activity Stream of Root Root",
"permaLink": "http://platform35.demo.exoplatform.org/profile/root"
} //optional
}
```

DELETE activity/:activityId.format

Description: Deletes an existing activity by its Id using the DELETE method. The deleted activity information will be returned in the JSON format.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity/:id.format
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e6f7g8h9i.json*

- *http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity/1a2b3c4d5e6f7g8h9i.json*

Supported Format: JSON **Requires Authentication**: true **Parameters**:

- **Required**
  - **id**: The Id of the existing activity.

Examples

- **JSON**

Request:

```
DELETE: http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e6f7g8h9i.json
```

Response:

```
{
"id": "1a2b3c4d5e6f7g8h9j",
"title": "Hello World!!!",
"appId": "",
"type": "exosocial:core",
"postedTime": 123456789, //timestamp
"createdAt": "Fri Jun 17 06:42:26 +0000 2011", //The Date follows ISO 8601
"priority": 0.5, //between 0.0 and 1.0, higher value => higher priority.
"templateParams": {},
"titleId": "",
"identityId": "123456789abcdefghi", //the identity id of the user who created this activity
"liked": true, //is liked (favorites) by this authenticated identity
"likedByIdentities": ["identityId1", "identityId2"],
"posterIdentity": {}, //optional
"comments": [{}, {}, {}], //optional
"totalNumberOfComments": 1234,
"activityStream": {
  "type": "user", // or "space"
  "prettyId": "root", // or space_abcde
  "faviconURL": "http://demo3.exoplatform.org/favicons/exo-default.jpg",
  "title": "Activity Stream of Root Root",
  "permaLink": "http://platform35.demo.exoplatform.org/profile/root"
```

```
} //optional
}
```

POST activity/destroy/:activityId.format

Description: Deletes an existing activity by its Id using the POST method. The deleted activity information will be returned in the JSON format. It is recommended to use the DELETE method, except the case that clients cannot make request via this method.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity/destroy/:id.format
```

For example: *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/destroy/1a2b3c4d5e6f7g8h9i.json*

http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity/destroy/1a2b3c4d5e6f7g8h9i.json

Supported Format: JSON **Requires Authentication**: true **Parameters**:

- **Required**
  - **id**: The Id of the existing activity.

Examples:

- **JSON**

Request:

```
POST: http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/destroy/1a2b3c4d5e6f7g8h9i.json
```

Response:

```
{
"id": "1a2b3c4d5e6f7g8h9j",
"title": "Hello World!!!",
```

```
"appId": "",
"type": "exosocial:core",
"postedTime": 123456789, //timestamp
"createdAt": "Fri Jun 17 06:42:26 +0000 2011", //The Date follows ISO 8601
"priority": 0.5, //between 0.0 and 1.0, higher value => higher priority.
"templateParams": {},
"titleId": "",
"identityId": "123456789abcdefghi", //the identity id of the user who created this activity
"liked": true, //is liked (favorites) by this authenticated identity
"likedByIdentities": ["identityId1", "identityId2"],
"posterIdentity": {}, //optional
"comments": [{}, {}, {}], //optional
"totalNumberOfComments": 1234, //if comments is required, the total number of comments
"activityStream": {
  "type": "user", // or "space"
  "prettyId": "root", // or space_abcde
  "faviconURL": "http://demo3.exoplatform.org/favicons/exo-default.jpg",
  "title": "Activity Stream of Root Root",
  "permaLink": "http://platform35.demo.exoplatform.org/profile/root"
} //optional
}
```

activity/:activityId/comments

GET activity/activity:id/comments.format

Description: Gets the comments on an activity.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity/:id/comments.format
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e/comments.json*
- *http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity/1a2b3c4e5e/comments.json*

Supported Format: JSON

Requires Authentication: true

Parameters: No

Examples:

- **JSON**

Request:

```
GET:  http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e/comments.json
```

Response: A list of comment objects in the JSON format.

```
{
  total: 10,
  comments: [
  {
    "id": "123456"
    "identityId": "12345abcde",
    "text": "Comment there!",
    "postedTime": 123456789,
    "createdAt": "Fri Jun 17 06:42:26 +0000 2011"
  },
  {
    "id" : "234567"
    "identityId": "12345abcde",
    "text": "Comment there 2!",
    "postedTime": 123456789,
    "createdAt": "Fri Jun 17 06:42:26 +0000 2011"
  }
  ]
}
```

activity/:activityId/comment

POST activity/:activityId/comment.format

Description: Posts a new comment on an existing activity. The poster of this comment is an authenticated identity.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity/:activityId/comment.format
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e/comment.json*
- *http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity/1a2b3c4e5e/comment.json*

Body data:

```
{
  "text": "My comment here!!!"
}
```

Supported Format:json

Requires Authentication: true

Parameters: No

Examples:

- **JSON**

Request:

```
POST:     http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e/comment.json
BODY: {"text": "My comment here!!!"}
```

Response: The created comment.

```
{
"id": "123456"
"identityId": "12345abcde",
"text": "My comment here!!!",
```

```
"postedTime": 123456789,
"createdAt": "Fri Jun 17 06:42:26 +0000 2011"
}
```

DELETE activity/:activityId/comment/:commentId.format

Description: Deletes an existing comment by its Id.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity/:activityId/comment/:commentId.format
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e/comment/123456.json*
- *http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity/1a2b3c4e5e/comment/123456.json*

Supported Format: JSON

Requires Authentication: true

Parameters: No

Examples:

- **JSON**

Request:

```
DELETE: http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e/comment/123456.json
```

Response: The deleted comment.

```
{
"id": "123456"
"identityId": "12345abcde",
```

```
"text": "My comment here!!!",
"postedTime": 123456789,
"createdAt": "Fri Jun 17 06:42:26 +0000 2011"
}
```

POST activity/:activityId/comment/destroy/:commentId.format

Description: Deletes an existing comment by its Id using the POST method. The deleted activity information will be returned in the JSON format. It is recommended to use the POST method, except the case that clients cannot make request via this method.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity/:activityId/comment/destroy/:commentId.format
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e/comment/destroy/123456.json*
- *http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity/1a2b3c4e5e/comment/destroy/123456.json*

Supported Format: JSON

Requires Authentication: true

Parameters: No

Examples:

- **JSON**

Request:

```
POST:  http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e/comment/destroy/123456.json
```

Response: The deleted comment

```
```

```
{
"id": "123456"
"identityId": "12345abcde",
"text": "My comment here!!!",
"postedTime": 123456789,
"createdAt": "Fri Jun 17 06:42:26 +0000 2011"
}
```

activity/:activityId/like

POST activity/:activityId/like.format

Description: Allows an authenticated identity to do the "like" action on an existing activity.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity/:activityId/like.format
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e/like.json*
- *http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity/1a2b3c4e5e/like.json*

Supported Format: JSON

Requires Authentication: true

Parameters: No

Examples:

- **JSON**

Request:

```
POST:  http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e/like.json
```

Response:

```
{
  "liked": "true"
}
```

DELETE activity/:activityId/like.format

Description: Allows an identity to remove his "like" action on an activity.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity/:activityId/like.format
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e/like.json*
- *http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity/1a2b3c4e5e/like.json*

Supported Format: JSON

Requires Authentication: true

Parameters: No

Examples:

- **JSON**

Request:

```
DELETE: http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e/like.json
```

Response:

```
{
```

```
  "liked": false
}
```

POST activity/:activityId/like/destroy.format

Description: Allows an identity to remove his "like" action on an activity. It is recommended to use the DELETE method, except the case that clients cannot make request via this method.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity/:activityId/like/destroy.format
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e/like/destroy.json*
- *http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity/1a2b3c4e5e/like/destroy.json*

Supported Format: JSON

Requires Authentication: true

Parameters: No

Examples:

- **JSON**

Request:

```
POST:  http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity/1a2b3c4d5e/like/destroy.json
```

Response:

```
{
  "liked": false
```

```
}
```

Activity Stream Resources

activity_stream/:identityId/user

GET activity_stream/:identityId/user/default.format

Description: Gets activities of a defined identity which can be an user identity or a space identity.

Url: s

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity_stream/{identityId}/user/default.{format}
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/user/default.json*
- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/user/default.json?limit=20*
- *http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity_stream/f92cd6f0c0a80137102696ac26430766/user/default.json*

Supported Format: JSON

Requires Authentication: true

Parameters:

- **Optional**
  - **limit**: The number of activities retrieved with the default value of 100. This input value must be less than or equal to its default value (100). The number of the returned results is actually less than or equal to the **limit** value.

Examples:

- **JSON**

Request

Response:

GET: http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity_stream/f92cd6f0c0a80137102696ac26430766/user/default.json

- **Response**


```
{
  "activities":[
    {
      "appId":null,
      "identityId":"f92cd6f0c0a80137102696ac26430766",
      "totalNumberOfComments":0,
      "liked":false,
      "templateParams":{

      },
      "postedTime":1309856902483,
      "type":"DEFAULT_ACTIVITY",
      "posterIdentity":null,
      "activityStream":null,
      "id":"f98e2d55c0a8013737dc8eabde05e7f7",
      "title":"Hello World 3",
      "priority":null,
      "createdAt":"Tue Jul 5 16:08:22 +0700 2011",
      "likedByIdentities":null,
      "titleId":null,
      "comments":null
    },
    {
      "appId":null,
      "identityId":"f92cd6f0c0a80137102696ac26430766",
      "totalNumberOfComments":0,
      "liked":false,
      "templateParams":{

      },
      "postedTime":1309856896732,
      "type":"DEFAULT_ACTIVITY",
      "posterIdentity":null,
      "activityStream":null,
      "id":"f98e16dec0a801377bfea51ee20d8b66",
```

```
    "title":"Hello World 2",
    "priority":null,
    "createdAt":"Tue Jul 5 16:08:16 +0700 2011",
    "likedByIdentities":null,
    "titleId":null,
    "comments":null
   },
   {
    "appId":null,
    "identityId":"f92cd6f0c0a80137102696ac26430766",
    "totalNumberOfComments":0,
    "liked":false,
    "templateParams":{

    },
    "postedTime":1309856884889,
    "type":"DEFAULT_ACTIVITY",
    "posterIdentity":null,
    "activityStream":null,
    "id":"f98de8aec0a801372dde36ca1cf69c6b",
    "title":"Hello World!!!",
    "priority":null,
    "createdAt":"Tue Jul 5 16:08:04 +0700 2011",
    "likedByIdentities":null,
    "titleId":null,
    "comments":null
   }
  ]
}
```


GET activity_stream/:identityId/user/newer/:activityBaseId.format

Description: Gets newer activities of a defined identity based on an activity Id.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity_stream/{identityId}/user/newer/{activityBaseId}.{format}
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/user/newer/0987654321.json*

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/user/newer/0987654321.json?limit=20*

Supported Format: JSON

Requires Authentication: true

Parameters:

- **Optional**
  - **limit**: The number of activities retrieved with the default value of 100. This input value must be less than or equal to its default value (100). The number of the returned results is actually less than or equal to the **limit** value.

Examples:

- **JSON**

Request:

```
GET: http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity_stream/f92cd6f0c0a80137102696ac26430766/user/newer/f98de8aec0a801372dde36ca1cf69c6b.json
```

Response:

```
{
  "activities":[
    {
      "appId":null,
      "identityId":"f92cd6f0c0a80137102696ac26430766",
      "totalNumberOfComments":0,
      "liked":false,
      "templateParams":{

      },
      "postedTime":1309856902483,
      "type":"DEFAULT_ACTIVITY",
      "posterIdentity":null,
      "activityStream":null,
      "id":"f98e2d55c0a8013737dc8eabde05e7f7",
      "title":"Hello World 3",
      "priority":null,
```

```
    "createdAt":"Tue Jul 5 16:08:22 +0700 2011",
    "likedByIdentities":null,
    "titleId":null,
    "comments":null
   },
   {
    "appId":null,
    "identityId":"f92cd6f0c0a80137102696ac26430766",
    "totalNumberOfComments":0,
    "liked":false,
    "templateParams":{

    },
    "postedTime":1309856896732,
    "type":"DEFAULT_ACTIVITY",
    "posterIdentity":null,
    "activityStream":null,
    "id":"f98e16dec0a801377bfea51ee20d8b66",
    "title":"Hello World 2",
    "priority":null,
    "createdAt":"Tue Jul 5 16:08:16 +0700 2011",
    "likedByIdentities":null,
    "titleId":null,
    "comments":null
   }
  ]
}
```


GET activity_stream/:identityId/user/older/:activityBaseId.format

Description: Gets older activities of a defined identity based on an activity Id.

Url:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity_stream/{identityId}/user/older/{activityBaseId}.{format}
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/user/older/0987654321.json*
- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/user/older/0987654321.json?limit=20*

Supported Format: JSON

Requires Authentication: true

Parameters:

- **Optional**
  - **limit**: The number of activities retrieved with the default value of 100. This input value must be less than or equal to its default value (100). The number of the returned results is actually less than or equal to the **limit** value.

Examples

- **JSON**

Request:

```
GET: http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity_stream/f92cd6f0c0a80137102696ac26430766/user/older/f98e16dec0a801377bfea51ee20d8b66.json
```

Response:

```
{
  "activities":[
    {
      "appId":null,
      "identityId":"f92cd6f0c0a80137102696ac26430766",
      "totalNumberOfComments":0,
      "liked":false,
      "templateParams":{

      },
      "postedTime":1309856884889,
      "type":"DEFAULT_ACTIVITY",
      "posterIdentity":null,
      "activityStream":null,
      "id":"f98de8aec0a801372dde36ca1cf69c6b",
      "title":"Hello World!!!",
      "priority":null,
      "createdAt":"Tue Jul 5 16:08:04 +0700 2011",
```

```
    "likedByIdentities":null,
    "titleId":null,
    "comments":null
   }
 ]
}
```

activity_stream/:identityId/feed

GET activity_stream/:identityId/feed/default.format

Description: Gets the latest activity stream feed of a defined user identity.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity_stream/{identityId}/feed/default.{format}
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/feed/default.json*
- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/feed/default.json?limit=20*

Supported Format: JSON

Requires Authentication: true

Parameters:

- **Optional**
  - **limit**: Specifies the number of activities to be retrieved. This number must be less than or equal to **100**. The value you pass to **limit** is a maximum number of activities to be returned. The actual number of activities you receive maybe less than **limit**. If **limit** is not specified, the default value will be 100.

Examples

- **JSON**

Request:

```
```

```
GET: http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity_stream/f92cd6f0c0a80137102696ac26430766/feed/default.json
```

-
  - **Response**

```
{
  "activities":[
    {
      "appId":null,
      "identityId":"f98f1197c0a80137528356921f2948c7",
      "totalNumberOfComments":0,
      "liked":false,
      "templateParams":{

      },
      "postedTime":1309856997920,
      "type":"exosocial:spaces",
      "posterIdentity":null,
      "activityStream":null,
      "id":"f98fa222c0a801373d2b094dcc48bf9b",
                                  "title":"<a  href={{{"}}}/socialdemo/private/classic/profile/demo{{{"}}}
 target={{{"}}}_parent{{{"}}}>Demo gtn</a> has joined the space.",
      "priority":null,
      "createdAt":"Tue Jul 5 16:09:57 +0700 2011",
      "likedByIdentities":null,
      "titleId":null,
      "comments":null
    },
    {
      "appId":null,
      "identityId":"f98f1197c0a80137528356921f2948c7",
      "totalNumberOfComments":0,
      "liked":false,
      "templateParams":{
      },
      "postedTime":1309856961044,
      "type":"exosocial:spaces",
      "posterIdentity":null,
      "activityStream":null,
      "id":"f98f121dc0a80137294aeb29d84f202e",
```

```
    "title":"john space was created by <a href={{{"}}}/socialdemo/private/classic/profile/john{{{"}}}
target={{{"}}}_parent{{{"}}}>John Anthony</a> .",
    "priority":null,
    "createdAt":"Tue Jul 5 16:09:21 +0700 2011",
    "likedByIdentities":null,
    "titleId":null,
    "comments":null
  },
  {
    "appId":null,
    "identityId":"f98ed67ec0a801370fc7afc98400df35",
    "totalNumberOfComments":0,
    "liked":false,
    "templateParams":{

    },
    "postedTime":1309856945880,
    "type":"exosocial:spaces",
    "posterIdentity":null,
    "activityStream":null,
    "id":"f98ed6f1c0a80137086908621ded6fe3",
  "title":"demo space was created by <a href={{{"}}}/socialdemo/private/classic/profile/demo{{{"}}}
target={{{"}}}_parent{{{"}}}>Demo gtn</a> .",
    "priority":null,
    "createdAt":"Tue Jul 5 16:09:05 +0700 2011",
    "likedByIdentities":null,
    "titleId":null,
    "comments":null
  },
  {
    "appId":null,
    "identityId":"f92cd6f0c0a80137102696ac26430766",
    "totalNumberOfComments":0,
    "liked":false,
    "templateParams":{

    },
    "postedTime":1309856902483,
    "type":"DEFAULT_ACTIVITY",
    "posterIdentity":null,
    "activityStream":null,
    "id":"f98e2d55c0a8013737dc8eabde05e7f7",
    "title":"Hello World 3",
    "priority":null,
```

```
    "createdAt":"Tue Jul 5 16:08:22 +0700 2011",
    "likedByIdentities":null,
    "titleId":null,
    "comments":null
  },
  {
    "appId":null,
    "identityId":"f92cd6f0c0a80137102696ac26430766",
    "totalNumberOfComments":0,
    "liked":false,
    "templateParams":{

    },
    "postedTime":1309856896732,
    "type":"DEFAULT_ACTIVITY",
    "posterIdentity":null,
    "activityStream":null,
    "id":"f98e16dec0a801377bfea51ee20d8b66",
    "title":"Hello World 2",
    "priority":null,
    "createdAt":"Tue Jul 5 16:08:16 +0700 2011",
    "likedByIdentities":null,
    "titleId":null,
    "comments":null
  },
  {
    "appId":null,
    "identityId":"f92cd6f0c0a80137102696ac26430766",
    "totalNumberOfComments":0,
    "liked":false,
    "templateParams":{

    },
    "postedTime":1309856884889,
    "type":"DEFAULT_ACTIVITY",
    "posterIdentity":null,
    "activityStream":null,
    "id":"f98de8aec0a801372dde36ca1cf69c6b",
    "title":"Hello World!!!",
    "priority":null,
    "createdAt":"Tue Jul 5 16:08:04 +0700 2011",
    "likedByIdentities":null,
    "titleId":null,
    "comments":null
```

```
  }
 ]
}
```

GET activity_stream/:identityId/feed/newer/:activityBaseId.format

Description: Gets the newer activity stream feed of an identity based on an activity Id.

URL:

http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity_stream/{identityId}/feed/newer/{activityBaseId}.{format}

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/feed/newer/0987654321.json*
- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/feed/newer/0987654321.json?limit=20*

Supported Format: JSON

Requires Authentication: true

Parameters:

- **Optional**
  - **limit**: The number of activities retrieved with the default value of 100. This input value must be less than or equal to its default value (100). The number of the returned results is actually less than or equal to the **limit** value.

Examples:

- **JSON**

Request:

GET: http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity_stream/f92cd6f0c0a80137102696ac26430766/feed/newer/f98ed6f1c0a80137086908621ded6fe3.json

Response:

```
{
  "activities":[
    {
      "appId":null,
      "identityId":"f98f1197c0a80137528356921f2948c7",
      "totalNumberOfComments":0,
      "liked":false,
      "templateParams":{

      },
      "postedTime":1309856997920,
      "type":"exosocial:spaces",
      "posterIdentity":null,
      "activityStream":null,
      "id":"f98fa222c0a801373d2b094dcc48bf9b",
                                  "title":"<a     href={{{"}}}/socialdemo/private/classic/profile/demo{{{"}}}
 target={{{"}}}_parent{{{"}}}>Demo gtn</a> has joined the space.",
"priority":null,
"createdAt":"Tue Jul 5 16:09:57 +0700 2011",
"likedByIdentities":null,
"titleId":null,
"comments":null
},
{
"appId":null,
"identityId":"f98f1197c0a80137528356921f2948c7",
"totalNumberOfComments":0,
"liked":false,
"templateParams":{
},
"postedTime":1309856961044,
"type":"exosocial:spaces",
"posterIdentity":null,
"activityStream":null,
"id":"f98f121dc0a80137294aeb29d84f202e",
"title":"john space was created by <a href={{{"}}}/socialdemo/private/classic/profile/john{{{"}}}
 target={{{"}}}_parent{{{"}}}>John Anthony</a> .",
      "priority":null,
      "createdAt":"Tue Jul 5 16:09:21 +0700 2011",
      "likedByIdentities":null,
      "titleId":null,
      "comments":null
```

```
    }
  ]
}
```

GET activity_stream/:identityId/feed/older/:activityBaseId.format

Description: Gets the older activity stream feed on an identity based on an activity Id.

URL:

http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity_stream/{identityId}/feed/older/{activityBaseId}.{format}

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/feed/older/0987654321.json*
- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/feed/older/0987654321.json?limit=20*

Supported Format: JSON

Requires Authentication: true

Parameters:

- **Optional**
  - **limit**: The number of activities retrieved with the default value of 100. This input value must be less than or equal to its default value (100). The number of the returned results is actually less than or equal to the **limit** value.

Examples:

- **JSON**

Request:

GET: http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity_stream/f92cd6f0c0a80137102696ac26430766/feed/older/f98e16dec0a801377bfea51ee20d8b66.json

Response:


```
{
  "activities":[
    {
      "appId":null,
      "identityId":"f92cd6f0c0a80137102696ac26430766",
      "totalNumberOfComments":0,
      "liked":false,
      "templateParams":{

      },
      "postedTime":1309856884889,
      "type":"DEFAULT_ACTIVITY",
      "posterIdentity":null,
      "activityStream":null,
      "id":"f98de8aec0a801372dde36ca1cf69c6b",
      "title":"Hello World!!!",
      "priority":null,
      "createdAt":"Tue Jul 5 16:08:04 +0700 2011",
      "likedByIdentities":null,
      "titleId":null,
      "comments":null
    }
  ]
}
```


activity_stream/:identityId/connections

GET activity_stream/:identityId/connections/default.format

Description: Gets activities of a defined identity's connections based on an activity Id.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity_stream/{identityId}/connections/default.{format}
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/connections/default.json*

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/connections/default.json?limit=20*

Supported Format: JSON

Requires Authentication: true

Parameters:

- **Optional**:
  - **limit**: The number of activities retrieved with the default value of 100. This input value must be less than or equal to its default value (100). The number of the returned results is actually less than or equal to the **limit** value.

Examples:

- **JSON**

Request:

```
GET: http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity_stream/f92cd6f0c0a80137102696ac26430766/connections/default.json
```

Response:

```
{
  "activities":[
    {
      "appId":null,
      "identityId":"f92cd31ac0a801373fbb2043e05d3312",
      "totalNumberOfComments":0,
      "liked":false,
      "templateParams":{
        "SENDER":"demo",
        "RECEIVER":"john",
        "RELATIONSHIP_UUID":"f9cff1ecc0a8013742f758cfaee8db70"
      },
      "postedTime":1309861232904,
      "type":"exosocial:relationship",
      "posterIdentity":null,
      "activityStream":null,
      "id":"f9d0410ac0a8013731e419f08bea9993",
```

```
      "title":"I am now connected with <a href={{{"}}}/socialdemo/private/classic/profile/demo{{{"}}}
 target={{{"}}}_parent{{{"}}}>Demo gtn</a>",
"priority":null,
"createdAt":"Tue Jul 5 17:20:32 +0700 2011",
"likedByIdentities":null,
"titleId":"CONNECTION_CONFIRMED",
"comments":null
},
{
"appId":null,
"identityId":"f92cd31ac0a801373fbb2043e05d3312",
"totalNumberOfComments":0,
"liked":false,
"templateParams":{
"SENDER":"demo",
"RECEIVER":"john",
"RELATIONSHIP_UUID":"f9cff1ecc0a8013742f758cfaee8db70"
},
"postedTime":1309861213060,
"type":"exosocial:relationship",
"posterIdentity":null,
"activityStream":null,
"id":"f9cff39fc0a8013716a77b1ce66864d7",
"title":"<a                                    href={{{"}}}/socialdemo/private/classic/profile/demo{{{"}}}
 target={{{"}}}_parent{{{"}}}>Demo gtn</a> has invited <a href={{{"}}}/socialdemo/private/classic/
profile/john{{{"}}} target={{{"}}}_parent{{{"}}}>John Anthony</a> to connect",
      "priority":null,
      "createdAt":"Tue Jul 5 17:20:13 +0700 2011",
      "likedByIdentities":null,
      "titleId":"CONNECTION_REQUESTED",
      "comments":null
    },
    {
      "appId":null,
      "identityId":"f92cd31ac0a801373fbb2043e05d3312",
      "totalNumberOfComments":0,
      "liked":false,
      "templateParams":{

      },
      "postedTime":1309856926090,
      "type":"DEFAULT_ACTIVITY",
      "posterIdentity":null,
      "activityStream":null,
```

```
    "id":"f98e898cc0a801372c958fbfe847e2d3",
    "title":"john activity 2",
    "priority":null,
    "createdAt":"Tue Jul 5 16:08:46 +0700 2011",
    "likedByIdentities":null,
    "titleId":null,
    "comments":null
   },
   {
    "appId":null,
    "identityId":"f92cd31ac0a801373fbb2043e05d3312",
    "totalNumberOfComments":0,
    "liked":false,
    "templateParams":{

    },
    "postedTime":1309856921353,
    "type":"DEFAULT_ACTIVITY",
    "posterIdentity":null,
    "activityStream":null,
    "id":"f98e770fc0a8013762a63bbd2c426726",
    "title":"john activity 1",
    "priority":null,
    "createdAt":"Tue Jul 5 16:08:41 +0700 2011",
    "likedByIdentities":null,
    "titleId":null,
    "comments":null
   }
  ]
}
```


GET activity_stream/:identityId/connections/newer/:activityBaseId.format

Description: Gets newer activities of a defined identity's connections based on an activity Id.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity_stream/{identityId}/connections/newer/{activityBaseId}.{format}
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/connections/newer/0987654321.json*
- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/connections/newer/0987654321.json?limit=20*

Supported Format: JSON

Requires Authentication: true

Parameters:

- **Optional**
  - **limit**: The number of activities retrieved with the default value of 100. This input value must be less than or equal to its default value (100). The number of the returned results is actually less than or equal to the **limit** value.

Examples:

- **JSON**

Request:

```
GET: http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity_stream/f92cd6f0c0a80137102696ac26430766/connections/newer/f9cff39fc0a8013716a77b1ce66864d7.json
```

Response:

```
{
  "activities":[
    {
      "appId":null,
      "identityId":"f92cd31ac0a801373fbb2043e05d3312",
      "totalNumberOfComments":0,
      "liked":false,
      "templateParams":{
        "SENDER":"demo",
        "RECEIVER":"john",
        "RELATIONSHIP_UUID":"f9cff1ecc0a8013742f758cfaee8db70"
      },
      "postedTime":1309861232904,
```

```
      "type":"exosocial:relationship",
      "posterIdentity":null,
      "activityStream":null,
      "id":"f9d0410ac0a8013731e419f08bea9993",
       "title":"I am now connected with <a href={{{"}}}/socialdemo/private/classic/profile/demo{{{"}}}
 target={{{"}}}_parent{{{"}}}>Demo gtn</a>",
"priority":null,
"createdAt":"Tue Jul 5 17:20:32 +0700 2011",
"likedByIdentities":null,
"titleId":"CONNECTION_CONFIRMED",
"comments":null
}
]
}
```


GET activity_stream/:identityId/connections/older/:activityBaseId.format

Description: Gets older activities of a defined user identity's connections based on an activity Id.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity_stream/{identityId}/connections/older/{activityBaseId}.{format}
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/connections/older/0987654321.json*
- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/connections/older/0987654321.json?limit=20*

Supported Format: JSON

Requires Authentication: true

Parameters:

- **Optional**
  - **limit**: The number of activities retrieved with the default value of 100. This input value must be less than or equal to its default value (100). The number of the returned results is actually less than or equal to the **limit** value.

Examples:

- **JSON**

Request:

```
GET: http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity_stream/f92cd6f0c0a80137102696ac26430766/connections/older/f98e898cc0a801372c958fbfe847e2d3.json
```

Response:

```
{
  "activities":[
    {
      "appId":null,
      "identityId":"f92cd31ac0a801373fbb2043e05d3312",
      "totalNumberOfComments":0,
      "liked":false,
      "templateParams":{

      },
      "postedTime":1309856921353,
      "type":"DEFAULT_ACTIVITY",
      "posterIdentity":null,
      "activityStream":null,
      "id":"f98e770fc0a8013762a63bbd2c426726",
      "title":"john activity 1",
      "priority":null,
      "createdAt":"Tue Jul 5 16:08:41 +0700 2011",
      "likedByIdentities":null,
      "titleId":null,
      "comments":null
    }
  ]
}
```

activity_stream/:identityId/spaces

GET activity_stream/:identityId/spaces/default.format

Description: Gets activities of spaces that the defined identity is the member or the manager.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity_stream/{identityId}/spaces/default.{format}
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/spaces/default.json*
- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/spaces/default.json?limit=20*

Supported Format: JSON

Requires Authentication: true

Parameters:

- **Optional**
  - **limit**: The number of activities retrieved with the default value of 100. This input value must be less than or equal to its default value (100). The number of the returned results is actually less than or equal to the **limit** value.

Examples:

- **JSON**

Request:

```
GET: http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity_stream/f92cd6f0c0a80137102696ac26430766/spaces/default.json
```

Response:

```
{
  "activities":[
    {
      "appId":null,
      "identityId":"f98f1197c0a80137528356921f2948c7",
      "totalNumberOfComments":0,
      "liked":false,
```

```
      "templateParams":{

      },
      "postedTime":1309856997920,
      "type":"exosocial:spaces",
      "posterIdentity":null,
      "activityStream":null,
      "id":"f98fa222c0a801373d2b094dcc48bf9b",
                                  "title":"<a       href={{{"}}}/socialdemo/private/classic/profile/demo{{{"}}}
 target={{{"}}}_parent{{{"}}}>Demo gtn</a> has joined the space.",
"priority":null,
"createdAt":"Tue Jul 5 16:09:57 +0700 2011",
"likedByIdentities":null,
"titleId":null,
"comments":null
},
{
"appId":null,
"identityId":"f98f1197c0a80137528356921f2948c7",
"totalNumberOfComments":0,
"liked":false,
"templateParams":{
},
"postedTime":1309856961044,
"type":"exosocial:spaces",
"posterIdentity":null,
"activityStream":null,
"id":"f98f121dc0a80137294aeb29d84f202e",
"title":"john space was created by <a href={{{"}}}/socialdemo/private/classic/profile/john{{{"}}}
 target={{{"}}}_parent{{{"}}}>John Anthony</a> .",
      "priority":null,
      "createdAt":"Tue Jul 5 16:09:21 +0700 2011",
      "likedByIdentities":null,
      "titleId":null,
      "comments":null
    },
    {
      "appId":null,
      "identityId":"f98ed67ec0a801370fc7afc98400df35",
      "totalNumberOfComments":0,
      "liked":false,
      "templateParams":{

      },
```

```
      "postedTime":1309856945880,
      "type":"exosocial:spaces",
      "posterIdentity":null,
      "activityStream":null,
      "id":"f98ed6f1c0a80137086908621ded6fe3",
    "title":"demo space was created by <a href={{{"}}}/socialdemo/private/classic/profile/demo{{{"}}}
 target={{{"}}}_parent{{{"}}}>Demo gtn</a> .",
      "priority":null,
      "createdAt":"Tue Jul 5 16:09:05 +0700 2011",
      "likedByIdentities":null,
      "titleId":null,
      "comments":null
    }
  ]
}
```


GET activity_stream/:identityId/spaces/newer/:activityBaseId.format

Description: Gets newer activities from all spaces that the defined user identity is the member or the manager based on a specified activity Id.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity_stream/{identityId}/spaces/newer/{activityBaseId}.{format}
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/spaces/newer/0987654321.json*
- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1-alpha1/portal/activity_stream/1234567890/spaces/newer/0987654321.json?limit=20*

Supported Format: JSON

Requires Authentication: true

Parameters:

- **Optional**
  - **limit**: The number of activities retrieved with the default value of 100. This input value must be less than or equal to its default value (100). The number of the returned results is actually less than or equal to the **limit** value.

Examples:

- **JSON**

Request:

```
GET: http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity_stream/f92cd6f0c0a80137102696ac26430766/spaces/newer/f98f121dc0a80137294aeb29d84f202e.json
```

Response:

```
{
  "activities":[
    {
      "appId":null,
      "identityId":"f98f1197c0a80137528356921f2948c7",
      "totalNumberOfComments":0,
      "liked":false,
      "templateParams":{

      },
      "postedTime":1309856997920,
      "type":"exosocial:spaces",
      "posterIdentity":null,
      "activityStream":null,
      "id":"f98fa222c0a801373d2b094dcc48bf9b",
                                  "title":"<a       href={{{"}}}/socialdemo/private/classic/profile/demo{{{"}}}
 target={{{"}}}_parent{{{"}}}>Demo gtn</a> has joined the space.",
"priority":null,
"createdAt":"Tue Jul 5 16:09:57 +0700 2011",
"likedByIdentities":null,
"titleId":null,
"comments":null
}
]
}
```

GET activity_stream/:identityId/spaces/older/:activityBaseId.format

Description: Gets the older activities of spaces that the defined user identity is the member or the manager based on a specified activity Id.

URL:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/activity_stream/{identityId}/spaces/older/{activityBaseId}.{format}
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1/portal/activity_stream/1234567890/spaces/older/0987654321.json*
- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1/portal/activity_stream/1234567890/spaces/older/0987654321.json?limit=20*

Supported Format: JSON

Requires Authentication: true

Parameters:

- **Optional**
  - **limit**: The number of activities retrieved with the default value of 100. This input value must be less than or equal to its default value (100). The number of the returned results is actually less than or equal to the **limit** value.

Examples:

- **JSON**

Request:

```
GET: http://localhost:8080/rest-socialdemo/private/api/social/v1-alpha1/socialdemo/activity_stream/f92cd6f0c0a80137102696ac26430766/spaces/older/f98fa222c0a801373d2b094dcc48bf9b.json
```

Response:

```
{
  "activities":[
```

```
  {
    "appId":null,
    "identityId":"f98f1197c0a80137528356921f2948c7",
    "totalNumberOfComments":0,
    "liked":false,
    "templateParams":{

    },
    "postedTime":1309856961044,
    "type":"exosocial:spaces",
    "posterIdentity":null,
    "activityStream":null,
    "id":"f98f121dc0a80137294aeb29d84f202e",
     "title":"john space was created by <a href={{{"}}}/socialdemo/private/classic/profile/john{{{"}}}
target={{{"}}}_parent{{{"}}}>John Anthony</a> .",
    "priority":null,
    "createdAt":"Tue Jul 5 16:09:21 +0700 2011",
    "likedByIdentities":null,
    "titleId":null,
    "comments":null
  },
  {
    "appId":null,
    "identityId":"f98ed67ec0a801370fc7afc98400df35",
    "totalNumberOfComments":0,
    "liked":false,
    "templateParams":{

    },
    "postedTime":1309856945880,
    "type":"exosocial:spaces",
    "posterIdentity":null,
    "activityStream":null,
    "id":"f98ed6f1c0a80137086908621ded6fe3",
   "title":"demo space was created by <a href={{{"}}}/socialdemo/private/classic/profile/demo{{{"}}}
target={{{"}}}_parent{{{"}}}>Demo gtn</a> .",
    "priority":null,
    "createdAt":"Tue Jul 5 16:09:05 +0700 2011",
    "likedByIdentities":null,
    "titleId":null,
    "comments":null
  }
 ]
```

```
}
```

Identity Resources

identity

GET identity/:identityId.format

Description: Gets the identity and its associated profile by the activity Id. **Url**:

```
http://{domain_name}/{rest_context_name}/private/api/social/{version}/{portal_container_name}/identity/:identityId.format
```

For example:

- *http://platform35.demo.exoplatform.org/rest/private/api/social/v1/portal/identity/123456789.json*
- *http://localhost:8080/rest-socialdemo/private/api/social/v1/socialdemo/identity/123456789.json*

Supported Format: JSON

Requires Authentication: true

Parameters: No

Examples:

- **JSON**

Request:

```
GET: http://platform35.demo.exoplatform.org/rest/private/api/social/v1/portal/identity/123456789.json
```

Response:

```
{
  "id" : "123456789",
  "providerId": "organization",
  "remoteId": "demo",
```

```
  "profile": {
    "fullName": "Demo Gtn",
    "avatarUrl": "http://platform35.demo.exoplatform.org/profile/avatar/demo.jpg"
  }
}
```

Notes

Identity Resources: **Identity Resources** has been supported to get an Id of identity based on providerId and remoteId. eXo Social 1.2.0-GA has not implemented this feature yet, however there is a workaround solution for that.

GET: {rest_context_name}/{portal_container_name}/social/identity/:username/id/show.json

Description: Gets the identity Id from a provided username.

Url:

```
http://{domain_name}/{rest_context_name}/{portal_container_name}/social/identity/{username}/id/show.json
```

For example:

- *http://platform35.demo.exoplatform.org/rest/portal/social/identity/demo/id/show.json*
- *http://localhost:8080/rest-socialdemo/socialdemo/social/identity/demo/id/show.json*

Supported Format: JSON

Requires Authentication: false

Parameters: No

Examples:

- **JSON**

Request:

```
GET: http://localhost:8080/rest-socialdemo/socialdemo/social/identity/demo/id/show.json
```

Response:

```
{"id":"f92cd6f0c0a80137102696ac26430766"}
```

Version Resources

version/latest

GET version/latest.format

Description: Gets the latest eXo Social REST services version. This version number should be used as the latest and stable version that is considered to include all new features and updates of eXo Social REST services.

Url:

```
http://{domain_name}/{rest_context_name}/api/social/version/latest.format
```

For example: ***http://platform35.demo.exoplatform.org/rest/api/social/version/latest.json*** *http://localhost:8080/rest-socialdemo/api/social/version/latest.json*

Supported Format: JSON

Requires Authentication: false

Parameters: No

For example:

- **JSON**

Request:

```
GET: http://platform35.demo.exoplatform.org/rest/api/social/version/latest.json
```

Response:

```
{"version": "v1-alpha1"}
```

- **XML**

Request:

```
GET: http://platform35.demo.exoplatform.org/rest/api/social/version/latest.xml
```

Response:

```
<version>v1-alpha1</version>
```

version/supported

GET version/supported.format

Description: Gets eXo Social REST service versions that are supported. This is for backward compatibility. If a client application is using an older eXo Social REST APIs version, all APIs of the version still can work. The array MUST have the latest to oldest order. For example, [v2, v1, v1-beta1], but not [v1, v2, v1-beta1].

Url:

```
http://{domain_name}/{rest_context_name}/api/social/version/supported.format
```

For example:

- *http://platform35.demo.exoplatform.org/rest/api/social/version/supported.json*
- *http://localhost:8080/rest-socialdemo/api/social/version/supported.json*

Supported Format: JSON, XML

Requires Authentication: false

Parameters: No

Examples:

- **JSON**

Request:

```

```

```
GET: http://platform35.demo.exoplatform.org/rest/api/social/version/supported.json
```

Response:

```
{"versions": ["v1-alpha1"]}
```

- **XML**

Request:

```
GET: http://platform35.demo.exoplatform.org/rest/api/social/version/supported.xml
```

Response:

```
<versions>
  <version>v1-alpha1</version>
</versions>
```

Public Javascript APIs

All references for Opensocial Javascript APIs can be viewed *here.* [http://wiki.opensocial.org/index.php?title=JavaScript_API_Reference]

Social JCR Structure

Overview

As any other eXo's products, eXo Social fully complies the JCR standard to store data (identity, profile, activity, space and relationship). The eXo Social JCR structure is organized to conform the data storage for the individual purpose of eXo Social. With this structure, it is easy for you to manage and access the data properly.

See the eXo Social JCR structure in the chart below:

The root node of Social workspace is */production/* which consists of two child nodes including *soc:providers*, and *soc:spaces*.

soc:providers

The *soc:providers* node is used to store the provider entities. In eXo Social, there are two built-in provider entities, including *organization*, and *space*. Its type is *soc:providers* that has child nodes of the *soc:providerdefinition* type.

soc:<provider*name*>

The <provider*name*> *parameter in the* soc:<provider_name> *node depends on the identity providers. eXo Social has two identity providers, including* OrgarnizationIdentityProvider*, and* SpaceIdentityProvider *that contain organization identities (users) and space identities respectively.*

The *soc:<provider_name>* node of the *soc:providerdefinition* type has the *soc:<identityRemoteId>* child nodes that have the *soc:identitydefinition* type.

The *soc:identitydefinition* node type has the following properties:

Property Name	Required Type	Mutiple	Description
soc:providerId	String	false	The provider Id is considered as a namespace for the remote Id.
soc:isDeleted	Boolean	false	Show that if the provider Id is deleted or not via the provider.
soc:remoteId	String	false	The local Id from a provider Id.

and has the following child nodes:

Child Nodes	Default Primary Type	Description
soc:profile	soc:profiledefinition	Store the detailed information of an identity.
soc:relationship	soc:relationship	Store all the relationships of an identity that is in connection with other identities.
soc:receiver	soc:relationship	Store all the relationships which contain an identity invited to connect by other identities.
soc:sender	soc:relationship	Store all the relationships which contain an identity

Child Nodes	Default Primary Type	Description
		inviting other identities to connect with himself.
soc:ignored	soc:relationship	Store all the relationships which contain an identity ignored by other identities.
soc:ignore	soc:relationship	Store all the relationships which contain an identity ignoring other identities.
soc:activities	soc:activitylist	Store all activities in the activity stream of an identity.
soc:spacemember	soc:spaceslist	Store all spaces of which an identity is a member.
soc:spacependingmember	soc:spaceslist	Store all spaces which an identity is pending for validation to join.
soc:spaceinvitedmember	soc:spaceslist	Store all spaces which an identity is invited to join.
soc:spacemanagermember	soc:spaceslist	Store all spaces of which an identity is a manager.

The *soc:relationship* node type has the child nodes of the *soc:relationshipdefinition* type that has the following properties:

Property Name	Required Type	Multiple	Description
soc:to	Reference	false	The sender identity. It refers to the *soc:identity_ node type.*
soc:status	String	false	The status of the relationship, including three values: PENDING, CONFIRMED, and IGNORED.
soc:reciprocal	Reference	false	Denotes if the relationship is one way or two ways. It refers to the*soc:relationshipdefinition_ node type.*
soc:from	Reference	false	

Property Name	Required Type	Multiple	Description
			The receiver identity. It refers to the *soc:identity_ node type.*
soc:createdTime	Long	false	The time when the relationship is created.

The *soc:spaceslist* node type contains the child nodes of the *soc:spaceref* type that has the following properties:

Property Name	Required Type	Multiple	Description
soc:target	Reference	false	Refer to a space entity.

The *soc:profiledefinition* node type contains the *soc:avatar* child node of the nt:file type and other child nodes of the *soc:profilexp* type.

The *soc:profiledefinition* node type has the following properties:

Property Name	Required Type	Multiple	Description
soc:parentId	String	false	The identity Id of the profile.
void-Url	undefined	true	The URL to access the profile of an identity.
void-email	undefined	true	The email of an identity in his the profile.
void-firstName	undefined	true	The first name of an identity in his profile.
void-fullName	undefined	true	The full name of an identity in his profile.
void-lastName	undefined	true	The last name of an identity in his profile.
void-username	undefined	true	The username of an identity in his profile.

The *soc:profilexp* node type has the following properties:

Property Name	Required Type	Multiple	Description
soc:position	String	false	The job position of an identity at an organization.

Property Name	Required Type	Multiple	Description
soc:company	String	false	The company where an identity works.
soc:skills	String	false	The work skills of an identity.
soc:description	String	false	The description of an identity's position at an organization.
soc:endDate	String	false	The date when an identity stops working at an organization.
soc:startDate	String	false	The date when an identity starts working at an organization.

The *soc:activitylist* node type has the child nodes of the *soc:activityyear*. The *soc:activitylist* node type has the following properties:

Property Name	Required Type	Multiple	Description
soc:number	Long	false	The number of activities in the activities list. The default value is set to 0.

The *soc:activityyear* node type has the child nodes of the *soc:activitymonth* type which has the same properties as its parent node type (*soc:activitylist*).

The *soc:activitymonth* node type has the child nodes of the *soc:activityday* type that has the same properties as its parent node type (*soc:activitymonth*).

The *soc:activityday* node type has the child nodes of the *soc:activity* type that has the following properties:

Property Name	Required Type	Multiple	Description
soc:isComment	Boolean	false	Specify if an activity is a comment or not. The default value is false, meaning that it is a normal activity.
soc:likes	String	true	The list of identity Ids who like the activity.
soc:priority	Float	false	A float number between '0' and '1'

Property Name	Required Type	Multiple	Description
			represents the relative priority level of an activity in relation to other activities from the same source.
soc:postedTime	Long	false	The number which specifies the time at which an activity took place in milliseconds since the epoch.
soc:title	String	false	The string which specifies the primary text of an activity.
soc:titleId	String	false	The title Id of an activity.
soc:body	String	false	The string which specifies the body template message Id in the gadget spec. The body is an optional extended version of an activity.
soc:bodyId	String	false	The body Id of an activity.
soc:type	String	false	The application Id which creates an activity.
soc:externalId	String	false	An optional string Id which is generated by the posting application.
soc:url	String	false	The URL to access an activity.
soc:appId	String	false	The application Id which creates an activity.
soc:identity	Reference	false	The identity whose activity stream contain an activity.

Property Name	Required Type	Multiple	Description
soc:posterIdentity	Reference	false	The identity who creates an activity.

The *soc:activity* node type has the *soc:params* child node of the *soc:activityparam* type and other child nodes of the *soc:activity* type.

The *soc:activityparam* node type has the following property:

Property Name	Required Type	Multiple	Description
	String	false	A map of key-values.

soc:spaces

The *soc:spaces* node is used to store data of eXo Social spaces. Its node type is *soc:spaces* that includes child nodes of the *soc:spacedefinition* type.

The *soc:spacedefinition* type has the following properties:

Property Name	Required Type	Multiple	Description
soc:type	String	false	The type of space which is used to run in the Classic or WebOS mode.
soc:membersId	String	true	The list of usernames which are members of a space.
soc:pendingMembersId	String	true	The list of usernames which are pending for validation to join a space.
soc:managerMembersId	String	true	The list of usernames which are managers of a space.
soc:invitedMembersId	String	true	The list of usernames which are invited to join a space.
soc:visibility	String	false	The space visibility: public, private, and hidden.
soc:name	String	false	The space name.
soc:avatarLastUpdated	Long	false	The last time when the avatar is updated.

Property Name	Required Type	Multiple	Description
soc:displayName	String	false	The display name of a space.
soc:registration	String	false	The space registration status: open, validation, and close.
soc:description	String	false	The description of a space.
soc:priority	String	false	The space priority level that is used to sort spaces in the spaces list. It contains three values: 1, 2 and 3. The smaller value has the higher priority level.
soc:app	String	false	The list of applications with portlet Id, application name, and its state (installed, activated, deactivated).
soc:groupId	String	false	The group associated with the corresponding space.
soc:url	String	false	The link to access a space.