# uPortal architecture overview

## 1. Presentation assembly

uPortal is a framework for an integrated delivery of content gathered from an assortment of information sources. The primary function of the framework is to provide efficient and flexible engine for assembling a presentation. Given a set of information sources (channels), and a recipe on how to arrange and frame them (stylesheets), uPortal framework coordinates the compilation of the final document.

The starting point for any presentation assembly is always an abstract organization of channels: the **user layout** document. The assembly process transforms the userLayout document in three major stages to obtain the final document in a desired markup language.

The first stage translates an abstract userLayo ut hierarchy into the structural terms of the final presentation. That translation is termed **structure transformation**, and its logic is defined by the structure stylesheet. For example, the structure stylesheet for the default “tab and column” presentation will translate abstract user layout structure into a structure of “tab” and “column” elements.
After the structure transformation, uPortal initiates rendering cycles of the channels that will be incorporated into the final presentation.

The second stage of the assembly process will translate the result of the structure transformation into a target markup language. This translation is termed **theme transformation**, and its logic is defined by the theme stylesheet. For example, default nested-tables theme transforms a document produced by the tab-column structure transformation (a structure of tabs, columns, etc.) into a set of nested HTML tables that visually resemble tabs and columns.
The content provided by each individual channel will be incorporated into the result of the structure transformation.

The final stage serializes the result of the theme transformation together with the channel content into a stream of characters according to the rules of the target markup language and media.

## 2. Stylesheets

Both theme and structure transformations are directed by the XSLT stylesheets. Each stylesheet registered with the system must be described by a configuration file (SDF – stylesheet description file). The details of how to register a stylesheet and its description with the uPortal are described in the maintenance document.

### *Structure stylesheet:*

Structure stylesheet defines an XSLT transformation of a userLayout document into elements corresponding to the structural elements of the desired presentation. Structure stylesheets must be able to accept input documents with both <layout> and <layout_fragment> root elements. The format (DTD) of the resulting document is left to the stylesheet author's discretion, granted the following restrictions are observed:

- The <channel/> elements in the resulting document must contain all of the attributes and child elements that they contained in the source userLayout document. In other words, XSLT can add attributes and, possibly, child nodes to the <channel/> elements when copying them from the userLayout, but none of the attributes or children of the <channel/> elements specified in the userLayout can be removed or overwritten.
- The <channel/> elements must appear in the resulting document if and only if the content of that channel will be incorporated into the final presentation. In other words, one of the goals of the structure transformation is to select a subset of channels from the userLayout document whose content will be necessary for compiling the final presentation[1].

In order to construct a rich and flexible presentation from an abstract userLayout document, the stylesheet usually needs extra information.
The simpler type of information is passed through parameters – name value pairs passed to the XSLT stylesheet. Parameters are declared in the stylesheet using the standard <xsl:param/> binding. Parameters must be described in the stylesheet description file (SDF) and the file must be registered with the uPortal. For example, a default tab-column structure XSLT stylesheet (tab-column.xsl) declares an "activeTab" parameter that keeps track of the tab that user is currently viewing.
In some cases the information needs to be associated with the elements of the userLayout and parameter variables are not sufficient. Structure stylesheets can declare (in their description file) attributes associated with userLayout's <folder> or <channel> elements. These are structure stylesheet-defined folder (channel) attributes. For example, tab-column.sdf declares a "width" attribute associated with <folder> elements of the userLayout, and interprets the value of that attribute as a width of a column for those folder elements that are translated into <column> elements.

The portal can persist the values of both parameters and attributes defined by the stylesheet. Parameters are passed to the stylesheet upon the initiation of the transformation. Folder and channel attribute values are merged into the userLayout document just prior to the transformation, in other words the stylesheet can expect the declared attributes to be in the source document.
The values of stylesheet-defined parameters and attributes can be changed through URL syntax (see "URL parameter syntax" section), or by interacting with the UserPreferences class directly (see "Internal object structure" and "Privileged channels" subsection in the next section).

[1] If the result of the structure transformation will include <channel/> elements whose content will not show up in the final presentation, this will degrade performance of the portal, for it will be spending time rendering these unnecessary channels. Hint: it is often that one would like to select only a few channels for rendering, but provide information about many other channels in the layout for the purposes of rendering navigational controls. In that case, make sure to translate elements describing these extra channels into something other then standard <channel/> elements.

The following parameters are always passed to the structure stylesheets:

- userLayoutRoot – an Id of a userLayout node defining a layout subtree that should be rendered. The intent is to provide an ability to "focus" on a particular part of the user layout. For example, default tab-column layout provides an ability to view a single channel in a "full screen" mode, where the selected channel occupies nearly all of the screen real-estate, with only header and footer being displayed. It is up to the stylesheet implementation to determine how the "focus" mode is presented, and what elements of the layout can be "focused".
- baseActionURL – a URL to which standard HTTP request parameters can be appended. This is provided so that stylesheets are able to construct targets that will pass request parameters back to the portal. The parameter is also passed to the theme XSLT stylesheets, where it is a lot more useful. Structure transformations rarely can incorporate URL elements since their resulting documents are expected to be independent of any particular markup language.

### *Theme stylesheet:*

Theme stylesheet is expected to transform an output produced by the structure transformation into a target markup language. The theme XSLT stylesheet must be able to understand the output produced by the structure transformation. Since these outputs usually vary significantly, the theme stylesheet must specify in its description file what structure transformation it can work with.
The restrictions on the output format are the same as in the case of a structure stylesheet (i.e. those concerning <channel/> elements).
Among other things, the stylesheet description file for theme stylesheets includes specification of target mime type, name of the serializer to be used to produce character stream representation and a type of a device that commonly supports targeted markup language.
Theme stylesheets can also define their own parameters and channel attributes. They can not, however, define folder attributes since the existence of folders can no longer be guaranteed after the structure transformation.
Same standard parameters (baseActionURL and userLayoutRoot) are passed to the theme XSLT. Stylesheet-defined channel attributes are also incorporated just prior to the transformation.

## 3. Channels

Channels are the unit sources of information for the uPortal. The framework orchestrates channel behavior to produce a coordinated output of the content provided by them. uPortal framework provides channels with the means to achieve their goals, but does not try to enforce how things are done. The bottom line is: uPortal does not try to be a container.
Administrative life of a channel consists of the following phases:

1. Creation. At this phase the code for the channel class and all of the necessary resource files are created and a **channel publishing document** (CPD file) is produced. CPD files provide descriptive information about the purpose of the channel, specify the channel Java class, describe configuration parameters, and

outline the workflow of the publishing and subscription process for this channel. The above tasks are usually carried out by the channel author. Upon the completion of this phase, the channel can be distributed to the uPortal installations.

2. Registration. When an administrator of a uPortal installation acquires a new channel, the channel has to be registered with the system. During the registration process, the portal is made aware of the channel's existence, and the channel is assigned channelTypeId.
3. Publishing. In order for a registered channel to become available for subscription by uPortal users, it has to go through a publishing process. During this process channel configuration is determined by assigning values to the channel's parameters[2]. Such channel configuration is assigned a channelPublishId. A published channel is then placed in taxonomy of channel categories.
4. Subscription. uPortal users can go through a subscription process to insert a published channel into their personal layouts. Subscription process finalizes channel configuration by assigning values to all of the necessary channel parameters. A subscribed channel is identified by a channelSubscribeId that is unique within the scope of the user's layout.

Internally, channels are Java objects implementing org.jasig.portal.IChannel interface[3]. Channels are stateful entities, and spend most of their lifetime in a rendering cycle. The rendering cycle begins with a call to a method that updates the state of a channel (IChannel.setRuntimeData()) and proceeds to a method that retrieves channel rendering in its current state (IChannel.renderXML()). Channels must be capable of presenting a rendering of their current state in a well-formed XML (a SAX stream as opposed to a string of characters or a character stream).

## *IChannel interface*

The org.jasig.portal.IChannel is the primary interface for the uPortal channels. The following guarantees are made by the uPortal about the call sequence of the IChannel methods:

1. setStaticData() method will be called once and only once, immediately after channel instantiation
2. Each rendering cycle will include a call to setRuntimeData() method, eventually followed by a call to renderXML() method.
3. receiveEvent() method can optionally be called during the rendering cycle, always prior to the renderXML() call.
4. getRuntimeProperties() method will be called during each rendering cycle before a call to renderXML() method, but after a call to setRuntimeData() method.

When prompted for content with the renderXML() method, channels are expected to provide a well-formed XML in a form of a SAX stream. The XML document produced

[2] Publisher may choose to allow some of the parameters to be modified or determined by the user, during a subscription stage.
[3] Alternatively org.jasig.portal.IMultithreadedChannel interface can be implemented

by a channel should be a fragment of a document constructed in the current markup language[4].

ChannelRuntimeData contains information about incoming request. Although uPortal does not allow regular channels to access to the HTTP request object directly, the HTTP parameters passed to the channel in the current HTTP request are available in the ChannelRuntimeData object along with a number of other pieces of information about the request object. uPortal uses context path of the incoming request URL to determine the channel to which request parameters are intended. Channels can construct self-addressing URL targets[5] by appending parameter lists to the baseActionURL provided to each channel in the ChannelRuntimeData.
By convention, uPortal reserves all parameter names that start with the "uP_" string.

uPortal does not specify how the channel content should be generated. It is common for channels to take XSLT-based approach in order to facilitate accommodation of multiple output formats.

## *IPrivileged*

Some channels require access to internal structures of the uPortal framework, explicit interaction with HTTP request and response objects and priority processing. Those channels implement org.jasig.portal.IPrivileged interface, and considered to be the "trusted" channels. The channels are passed PortalControlStructures object at each rendering cycle, that provides them with direct access to the core framework classes.

uPortal guarantees that if parameters in a given request are addressed to a privileged channel, the setRuntimeData() method of that channel will be invoked (and will return) before any of the other channels in the current session begin their rendering cycle[6].

uPortal assumes that a privileged channel will not attempt to delete itself from the user's layout when that channel is the current rendering root.

## *ICacheable*

Channels can implement org.jasig.portal.ICacheable interface if they wish to delegate content caching to the uPortal framework. In most cases caching allows to achieve significant performance boosts. The interface requires channels to be able to produce keys uniquely corresponding to the channel states, and provides a facility for a run-time validation of the existing keys by the channel. It is highly recommended that high-exposure channels implement this interface with the most general set of keys possible.

[4] For example, in the case of HTML, the channels are expected to output an HTML document without either <html>. <body> or <head> tags.
[5] For example, if baseActionURL is used as an "action" attribute in the HTTP form element, all of the parameters submitted by the form will be routed to the channel that has constructed the form, in the ChannelRuntimeData object.
[6] This allows privileged channels to modify userLayout, UserPreferences and other instantiated channels before the rendering assembly starts, thereby being able to control every aspect of the response assembly.

### *IMultithreadedChannel*

In some cases the overhead of a channel instance can be significant. Channels can choose to have a single instance of their class generate content for all occurrences of that channel in the user's layouts. This can be done by implementing org.jasig.portal.IMultithreadedChannel interface instead of a regular IChannel interface. The assumptions and guarantees for each delineated session of the multithreaded channels are the same as for a regular IChannel.

### *Inter-channel Communication*

uPortal provides the means for channels to locate and establish communication with the channels in the same user layout. The JNDI context obtained from the ChannelStaticData object, contains a "/channel-ids" with subcontexts corresponding to the functional names (fname) of the channels in the user's layout. The channelSubscribeId is bound to each of the sub-contexts defined by the functional names. This provides the means to determine channelSubscribeId given channel's standardized functional name.
Once the channelSubscribeId is known, communication to the desired channel can be accomplished either through URL parameter passing (see "URL parameter syntax" appendix), or by locating a channel-bound sub-context in the "/channel-obj" branch of the root JNDI context. Each sub-context of the "/channel-obj" branch is named with the channelSubscribeId of the channel owing that sub-context. Channels owning these sub-contexts can bind arbitrary objects in that space[7].

### *Channel Serivces*

Entities providing common functionality for the channels in the uPortal, can be registered and bound in the JNDI "/services" branch. That branch will be available in the JNDI context passed to every channel in the ChannelStaticData object. Services can be used to provide a more structured, mediated way of implementing channel communication.

### *Workers*

uPortal framework attempts to limit the abilities of individual components (such as channels) to control the overall output of the servlet. In a number of cases it is desirable that some module to gain complete control over the HTTP response, while remaining an integral part of the uPortal. For example, a particular channel might require to serve a binary stream (file, video) in the response.
Such functionality is accomplished through use of workers. A URL targeting a particular worker is constructed by using ChannelRuntimeData.getWorkerActionURL() value in the same way one would use a "baseActionURL" string. Workers are configured in the worker.properties file, and the 2.0 release includes an implementation of a file download worker.

## 4. Internal object structure

The internal organization of the uPortal framework is a continuously evolving entity, however the following major classes are likely to retain their responsibilities:

---

[7] For example a channel can bind an event-generating object in its private sub-contexts and allow other channels register as listeners.

- PortalSessionManager. This is an entry point into the uPortal. It is a J2EE servlet, conforming to the specification version 2.3. It is responsible for identifying and managing incoming HTTP requests and dispatching them to corresponding UserInstance objects.
- UserInstance is an object that encompasses all of the information for an individual user. It contains UserLayoutManager and ChannelManager objects for the user's session. It also bares the responsibility of orchestrating the rendering pipeline.
- UserLayoutManager maintains the userLayout document and UserPreference objects. It mediates any changes made to the layout or preferences for the current user session.
- ChannelManager maintains instances of channels for the current session. It is responsible for distributing runtime information to the channels and spawning ChannelRenderer objects. ChannelManager also maintains channel caches.
- ChannelRenderer objects are responsible for driving rendering cycles of individual channels.
- JNDIManager is responsible for initializing and maintaining portal JNDI context.
- PropertiesManager relates portal configuration information to other classes

## 5. Integration with the existing infrastructures

uPortal was devised as a framework that can accommodate a wide range of existing infrastructure set-ups. Each area of external integration is separated by an interface (see "Published interfaces" appendix), and the default distribution provides one or more reference implementations for these interfaces.

### *Authentication and Directory information subsystems*

The Authentication components of the uPortal handle the user login to uPortal. The major players in the reference implementation are the following.

- The Login channel handles the display of the form for entering username and password. The default security provider in the reference implementation is SimpleSecurityContext which requires a username and password. The rest of the authentication components are designed to be extremely flexible and can use non-password based authentication. In that case the Login Channel would not be required.

- The AuthenticationServlet handles the request and post data generated by the user submitting the login form. After authentication is completed (or fails) AuthenticationServlet redirects back to the main portal infrastructure.

- The Authentication service is called by AuthenticationServlet to (1) perform authentication, (2) acquire directory information for an authenticated user and (3) establish the uPortal specific identity for this user including the initial layout. The Authentication service can handle multiple principals and credentials.

1. The Security Provider interface is invoked by the Authentication service to validate credentials for a given principal (user). The reference implementation includes several example implementations of security providers and is set up by default to use SimpleSecurityContext which validates the password against the md5 password hash stored in the reference implementation's RDBMS. This interface is described in detail in the document pointed to from the developer documentation section of the uPortal website. All the properties governing the behavior of the security context interface are specified in security.properties EXCEPT for org.jasig.portal.security.provider.ChainingSecurityContext.stopWhenAuthenticated which was added to the portal.properties file.

2. The PersonDirectory service is called by the Authentication service to acquire directory attributes for the authenticated user. The PersonDirectory service provided in the reference implementation uses any number of jndi and/or jdbc sources to retrieve attributes and map them into the uPortal person object. The mapping of attribute names is determined by the PersonDirs.xml file. The channel, CPersonAttributes, is provided as an example of how to display the person attributes of the current user. [Comments in the PersonDirs.xml file explain the syntax for specifying the sources to query.]

3. The IUserIdentityStore interface is the mechanism for retrieving the unique identifier for the current user in the portal. When the property org.jasig.portal.services.Authentication.autoCreateUsers is set to true this component will automatically create all necessary data for a new portal user based on a template user. The template user is specified either by mapping a directory attribute to uPortalTemplateUserName. If no attribute is of that name is mapped the template user name comes from the org.jasig.portal.services.Authentication.defaultTemplateUserName property.

- Md5passwd
  The stand alone program md5passwd is provided as a rudimentary way to add a new user to the simpleSecurityContext or change a password. The data affected is in the UP_PERSON_DIR table of the uPortal rdbms. This is really a stand-in for a real source of directory information such as LDAP.

After the user is logged in the security context and person objects are available to channels via channelStaticData.

## *Authorization*

**Permissions**

The unit of authorization in the portal is the *Permission*, represented by the type IPermission. A Permission is granted by an *owner*, typically a channel, to a *Principal*. A Permission gives its Principal the right to perform some *activity*. In addition, a Permission can be further modified with a *target*, a *type*, and *effective* and *expiration* dates. The Permission activity and target are simply tokens that the owner interprets as it sees fit.  At present the Permission is simply a data holder with attributes for owner, principal type, principal key, activity, target, type, effective date, and expiration date.

**Authorization and Groups**

A Principal can be any portal entity including a group (an IEntityGroup) or a group member.  The authorization system works with the groups system to retrieve the Principal's group memberships.  As a result, adding an entity to a group allows the entity's Principal to inherit Permissions granted to the group.

**The Authorization Service**

The authorization design looks at authorization from three different points of view:  the portal *user*, the Permission *owner*, and the Permission *maintainer*.  Each point of view is represented by a separate type, an alternate façade for the authorization service.  The types are:  IAuthorizationPrincipal (user), IPermissionManager (owner) and IUpdatingPermissionManager (maintainer).  The authorization service is a factory for these façades.

A client wishing to do authorization asks the service for the appropriate type.  Based on its Permissions, the client gets (or does not get) the type it has requested, constrained by the client's Permissions.

**IAuthorizationPrincipal**

An IAuthorizationPrincipal represents a particular principal and can answer questions about what that principal can do.  A principal can be a group member, so in addition to its own Permissions, it inherits the Permissions of its group(s).

**IUpdatingPermissionManager**

An IUpdatingPermissionManager also represents a Permission owner, but at maintenance time when the owner is writing Permissions, rather than runtime when the owner is evaluating them.  It inherits methods for retrieving IAuthorizationPrincipals and Permissions from IAuthorizationManager and in addition, has methods for updating the IPermission store.

While IAuthorizationPrincipal and IPermissionManager will want to cache aggressively to optimize performance, an IUpdatingPermissionManager will ignore cached Permissions so that it can guarantee synchronization with the IPermission store.

### *Storage*

uPortal defines store interfaces for all of the data required to manage the portal. A normalized relational database reference implementation of these interfaces is provided. Interfaces make an implicit assumption that all of the data served by the interface implementation is self-consistent. (For example, an implementation would not return a user profile that is using a non-existing core stylesheet).

## 6. Guest users

uPortal provides performance optimizations for guest users. There can be multiple guest user accounts on the system[8].

[8] Guest user is determined according to the IPerson.isGuest() return value.

For optimal performance and maintainability, it is advised that guest users have the following characteristics:

- The userLayout and preferences for the user change very infrequently
- The user has many concurrent sessions on the system
- The user is not authenticated

uPortal does not allow guest users to persist any changes on the system. Default portal installation assumes that guest users are never authenticated and if the user does, in fact, authenticate with the uPortal, that session will be treated as a regular user[9].

## Appendix A: URL parameter syntax

uPortal framework reserves the use of HTTP request parameters, whose names start with the "`uP_`" substring. In this space, uPortal defines a parameter syntax that can be used by either channels or theme/structure stylesheets to communicate a limited set of events to the core uPortal classes.

Notation:

- syntax is given in a standard regular expression form.
- `{x}` denotes a value of some variable `x`
- `a & b` denotes that two parameter-value pairs ('`a`' and '`b`') are being passed with the request.
- all other words are literal strings

Accepted HTTP request parameter syntax:

Activate uPortal channel actions (help, about and edit):

- `(uP_help_target={channelSubscribeId})*`
- `(uP_about_target={channelSubscribeId})*`
- `(uP_edit_target={channelSubscribeId})*`

`Remove a particular target (folder or a channel):`

- `(uP_remove_target=({folderId}|{channelSubscribeId}))*`

`Set a detach target`[10] `(folder or channel):`

- `(uP_detach_target=({folderId}|{channelSubscribeId}))?`

`Persist current state of the user layout and/or preferences:`

- `(uP_save=(all|preferences|layout))?`

`Tell structure/theme transformation to focus on a particular channel:`

- `(uP_root=(me|{channelSubscribeId})?`

`Set the value of structure or theme stylesheet parameter:`

- `(uP_sparam={pName} & {pName}={pValue})*`
- `(uP_tparam={pName} & {pName}={pValue})*`

`Set value(s) of structure folder attribute(s):`

[9] This allows administrators to log in as "guest" users in order to maintain these accounts.
[10] If detach target is set to something other then root, a <layout_fragment> containing the target node will be processed with the structure transformation instead of the entire layout.

- `(uP_sfattr={attName} & {attName}_folderId={folderId} & {attName}_value={attValue})*`

`Set value(s) of structure channel attribute(s):`

- `(uP_scattr={attName} & {attName}_folderId={folderId} & {attName}_value={attValue})*`

`Set value(s) of theme channel attribute(s):`

- `(uP_tcattr={attName} & {attName}_channelId={channelSubscribeId} & {attName}_value={attValue})*`

## Appendix B: published interfaces

The uPortal framework defines a number of core interfaces that characterize the interactions between the framework and the outside environment. These interfaces will be carefully evolved in the future releases to allow for a gradual change (through usual deprecation mechanisms), or, if the situation allows, kept the same[11].

The following interfaces can be considered published in the uPortal 2.2 release:

- org.jasig.portal.IChannel
- org.jasig.portal.ICacheable
- org.jaisg.portal.IMultithreadedChannel
- org.jsaig.portal.ICharacterChannel
- org.jasig.portal.IPrivileged
- org.jasig.portal.IUserLayoutStore
- org.jasig.portal.IChannelRegistryStore
- org.jasig.portal.IUserIdentityStore
- org.jasig.portal.IUserPreferencesManager
- org.jasig.portal.IBasicEntity
- org.jaisg.portal.groups.IGroupService
- org.jasig.portal.groups.IGroupMember
- org.jasig.portal.groups.IEntityGroup
- org.jasig.portal.groups.IEntity
- org.jasig.portal.groups.IEntityNameFinder
- org.jasig.portal.groups.IEntitySearcher
- org.jasig.portal.layout.IUserLayout
- org.jasig.portal.layout.IUserLayoutManager
- org.jasig.portal.layout.IUserLayoutNodeDescription
- org.jasig.portal.layout.restrictions.IUserLayoutRestriction
- org.jasig.portal.security.ISecurityContext
- org.jasig.portal.security.IOpaqueCredentials
- org.jasig.portal.security.IPerson
- org.jasig.portal.security.IPermission
- org.jasig.portal.security.IAuthorizationService

[11] No such promises are made regarding non-published interfaces and classes. They might disappear overnight ☺

- org.jasig.portal.security.IAuthorizationPrincipal

## Appendix C: local and remote resource addressing

Since uPortal is normally deployed as a WAR file, local resources (such as files) are accessed through class loaders. uPortal provides a special utility class to assist with resource loading: org.jasig.portal.utils.ResourceLoader.

There are three ways to identify the resource:

### *A full URI with protocol*

In this case, a complete, WAR-file independent, URI is specified.
Examples:

- http://www.interactivebusiness.com/publish/ibs.rss
- file:///users/home/kweiner (UNIX)
- file:/C:/Projects/uPortal/webpages/rss/news.rss (Windows)

### *An absolute CLASSPATH URI*

In this case a path to the resource, relative to one of the CLASSPATH elements is specified. A path must begin with a forward slash. The ResourceLoader will search each CLASSPATH element root for the resource, using the specified path.
Examples:

- /org/jasig/portal/layout/tab-column/tab-column.xsl
- /properties/portal.properties
  (if org.jasig.portal.MyClass uses ResourceLoader to look for "/properties/portal.properties", it will be found at "WEB-INF/classes/properties/portal.properties")

### *A package-relative CLASSPATH URI*

In this case a path to the resource is specified relative to the package name of the calling class. The system class loader will search each CLASSPATH element root for the resource using the path specified appended to a path that matches the java package name of the class that is requesting the resource.
A path must not begin with a forward slash.
Examples:

- CGenericXSLT/RSS/RSS.cpd
- myStylesheet.xsl
  (if org.jasig.portal.MyClass uses ResourceLoader to look for "myStylesheet.xsl", it will be found at "WEB-INF/classes/org/jasig/portal/myStylesheet.xsl")