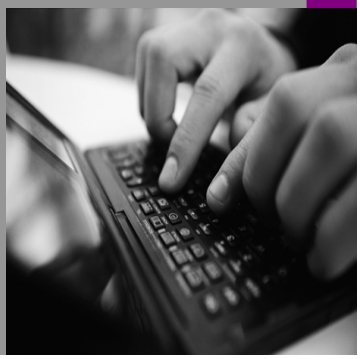


How-to Guide

SAP NetWeaver 2004

How To... Automate Content Creation via XML (XML Content and Actions)

Version 3.5 – August 2007

Applicable Releases:

SAP NetWeaver 2004, SP Stack 21

Revision History

Date	Version Number	NetWeaver Release	Description
July 2005	2	2004 SPS13	Adds the following <Context> elements: • Translation worklist • Related Items • System Aliases Adds the following <Action> elements: • Running Script within Script Shortens introductory chapters 1 and 2. Entire document re-edited.
August 2005	2.1	2004 SPS13	Adds handler for modifying permissions (<ACL> element). Adds revision history.
December 2005	3.0	2004 SPS15	Adds export tool.
June 2006	3.1	2004 SPS17	Describes how to add meta-attributes.
May 2007	3.2	2004 SPS19	Re-edited.
June 2007	3.3	2004 SPS20	Re-edited.
August 2007	3.4	2004 SPS21	Adds limitation that multi-value attributes are not supported.
August 2007	3.5	2004 SPS21	Adds support for specifying XML file encoding.

Contents

1 XML Content and Actions

Purpose

The portal enables administrators to write and import an XML file in order to automate the creation of portal semantic objects (such as iViews, pages and systems) and to perform actions (such as assigning roles or deleting content).

The process enables the creation of mass content without the use of standard portal wizards and editors. In addition, advanced users can perform batch operations and make pinpoint modifications within a large content base.

> **Note:** In previous versions of the portal, a service known as the *Generic Creator* was available as an SAP internal tool. The service has been rewritten as the XML Content and Actions feature and is now officially released to customers as a portal service and iView.

Creating Valid XML

The imported XML file must adhere to the specifications described in this document. The XML can be coded in a number of ways, including using scripts that transform Microsoft Excel or text documents to XML. Such services are not supplied by SAP.

You can also build a template for the XML file by creating an XML file based on existing content and then editing this file. The portal provides an export tool that creates an XML file from existing content, that is, it creates a file that you could then import into a portal to re-create the content. You can use the export tool to create template XML files.

The XML file is imported via a dedicated iView, which is assigned by default to the standard system administration role. For instructions on using the iView, see *Exporting/Importing Content and Actions* on page 35.

The portal parses the XML and generates the specified content and performs the specified actions.

Imported XML files can execute any number of actions in the portal, including overwriting and deleting existing content. Running an incorrect XML file may cause permanent damage to the portal. It is highly recommended to perform test runs initially on a non-production portal or on test content before using it in a live environment.

It is recommended to restrict access to the iView to administrators trained to use it.

XML Character Encoding

By default, the character encoding of all imported XML files is assumed to be UTF-8.

If you want to import files with a different encoding, specify the encoding in a property of the XML Content and Actions service. The property is located at:

- **Application:** `com.sap.portal.ivs.init`
- **Service:** `genericcreator`
- **Property:** `XML.file.encoding`

Navigate to *System Administration* → *System Configuration* → *Service Configuration* to update service properties.

Restart the service.

Constraints/Limitations

- The portal's transport (import/export) mechanism should be used to move content from one portal to another. The transport mechanism also provides additional functionalities, such as multi-language support, as well as the transport of applications and not just PCD content.
- The portal does not include an editor for viewing, editing, or validating the XML before it is imported.
- The creation and export of multi-value attributes are not supported. This limitation also affects OBN support, specifically, that iViews that are OBN targets cannot be exported.

1.1 Architecture

This section describes the major components of the XML Content and Actions feature, as well as the internal process flow that occurs when an XML file is imported.

1.1.1 Key Components

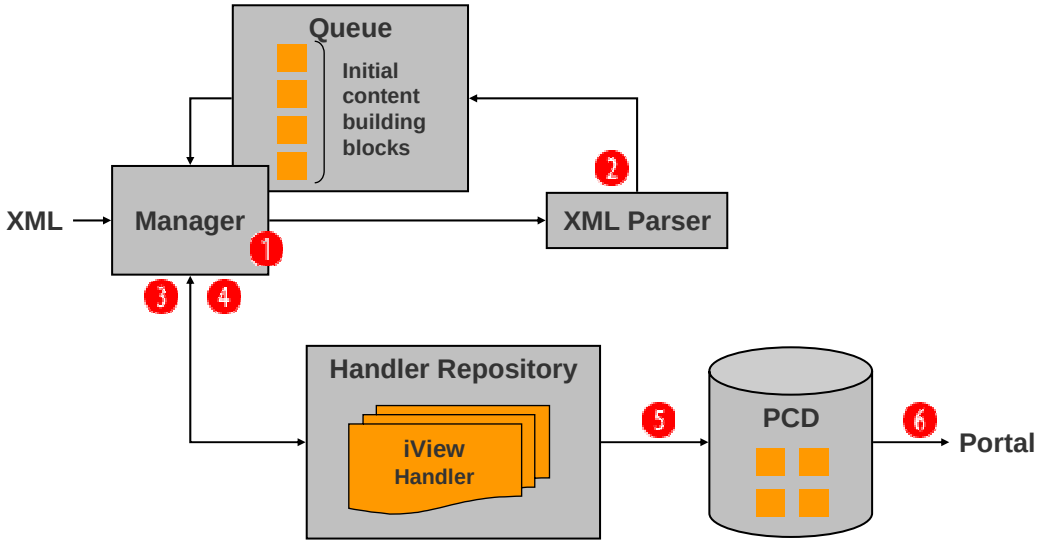
The following are the key components involved in creating content and performing actions:

Component	Function
XML Content and Actions export tool iView	• Creates an XML file based on selected content. This file is a well-formed and valid XML file that can be imported to a portal to create the same content.
XML file (created by an administrator)	• Specifies the objects to be created, updated and deleted. • Specifies the actions to be performed.
XML Content and Actions import tool iView	• Imports the XML file to the portal and passes it to the Generic Creator engine for processing.
Generic Creator engine	• Parses the XML. • Associates each XML block with the appropriate handler. • Manages the process of creating content and performing actions, including error handling.
Handlers	• Each handler is responsible for creating, modifying or deleting a specific semantic object, or for performing a specific action.

1.1.2 Process Flow

The following describes the process flow when importing an XML file via the import tool iView:

1. The XML is checked to make sure that it is well formed. If not, the process is aborted.
2. The XML is analyzed to determine what objects need to be created and what actions need to be performed. A set of building blocks is written into an execution queue.
3. Required handlers are loaded.
4. Handlers check each building block to make sure it can be executed. If at least one object or action in the queue cannot be executed, the entire process is aborted without writing anything to the PCD.
5. Handlers write the objects to the PCD and perform the actions specified in the XML.
6. A report on the results of the import is displayed in the import tool iView.



2 Workflow for XML Content and Actions

Purpose

This section describes the typical workflow for creating content and performing actions.

Prerequisites

- You need to plan what semantic objects to create and actions to perform.

  One method is to create a text file or Excel spreadsheet to list the required objects and actions, and then to run a script that generates XML from the text or Excel file.

  You may instead want to create objects in the portal, and then export the content to XML, and use this XML as a template for the XML file to be imported.

- Before importing the XML, all portal components on which objects defined in the XML file are based must already exist in the portal, including page layouts.

Process Flow

1. Create a well-formed and valid XML file according to the general requirements of the XML parser, as described in Section 3, *XML Elements and Attributes*, and the specific requirements of the handlers that you are using, as described in Section 4, *Available Semantic Objects and Actions*.

   > **Note:** You can also create an XML file by exporting content from the portal to XML and then editing this file. For more information on exporting content, see Section 5.1, *Exporting Content*.

2. Import the XML file via the *XML Content and Action* import tool in the portal, as described in Section 5.2, *Importing Content and Actions*.

   The tool is available in the portal at *System Administration* → *Transport* → *XML Content and Actions* → *Import*.

3. Review the results of the import in the user interface of the *XML Content and Action* import tool. The import report is described in Section 5.2, *Importing Content and Actions.*

4. Check and test the content in the portal.

3 XML Elements and Attributes

This section describes the schema for XML files parsed by the Generic Creator engine. The file has the following sections:

- **General Details:** Defines general details and configuration settings for the XML import. These are defined by attributes in the `[GenericCreator]` root element, as described in *XML Element: [GenericCreator]* on page 6. This section is mandatory.
- **Global Parameters:** Defines global properties and values that can be used throughout the XML file, as described in *XML Element: [Property]* on page 9.
- **[Context] or [Action] Element Blocks:** Defines either a semantic object to create or modify (`[Context]` element) or an action to perform (`[Action]` element), as described in *XML Element: [Context]* on page 10 and *XML Element: [Action]* on page 15.

3.1 XML Element: [GenericCreator]

The `[GenericCreator]` element must be the root element. It configures the XML file and the behavior of the XML parser.

Definition

The following is the format for the `[GenericCreator]` element:

```
<GenericCreator author="<author_name>"
    version="<version_and_description>" mode="<mode1>, <mode2>"
    report.level="<report_level>" ignore="<ignore_mode>"
    default.locale="<locale_ID>" createMode="<overwrite_mode>">
```

The following describes the `[GenericCreator]` element attributes:

Attribute	Mandatory	Description
`author`	No	Specifies the name of the author of the XML file. The author does not have to be defined as a user in the portal. This attribute has no effect on the portal or the XML import.
`ignore`	Yes	Specifies whether an XML block is executed. This attribute can also be applied to `[Context]` and `[Action]` elements. This attribute applies to each sub-block within the block that specifies it, unless the sub-block overrides the attribute value. For example, if the value in the `[GenericCreator]` element is **`true`**, then all XML blocks are skipped, unless `ignore` in a specific XML block is **`false`**. The following are valid values: • **`true`**: The block is not executed. • **`false`**: The block is executed (default). See also *Executing Specific XML Blocks* on page 34.

`report.level`	Yes	Specifies which messages are reported after an XML file has been imported. The following are valid values (default report levels implemented by the standard handlers): 1. `debug` 2. `info` 3. `warning` 4. `success` 5. `fail` Results are displayed from the selected report level and higher; for example, if `warning` is defined as the report level, then results of type `warning`, `success` and `fail` will also be displayed. If `debug` is defined, then all result types will be displayed.
`mode`	Yes	Specifies the mode for content creation. The following are valid values: • `clean`: Objects defined within `[Context]` elements are removed from the PCD. Most actions defined by elements of type `[Action]` are not performed, though each action can define an alternate action for this mode. • `execute`: Objects defined within `[Context]` elements are created in the PCD. Actions defined by `[Action]` elements are performed. The value of the `createMode` attribute (see below) determines how the `execute` mode is performed. If you specify more than one value – separated by commas – the script is executed once in the first mode, once in the second mode, and so on.

`createMode`	Yes	Specifies what to do when the XML defines an object that already exists in the PCD. Valid only when the `mode` attribute is `execute`. This attribute can also be applied to `[Context]` elements. The following are valid values: • `1`: If the object exists, then **do nothing** (default). • `2`: If the object exists, then **replace** the entire existing object and its properties with the new one. • `3`: If the object exists, then **update** only the properties that are declared in the XML document. In other words: (i) if the XML defines properties that already exist for the existing object, they are updated; (ii) if the XML defines new properties, they are added to the existing object; and (iii) if the existing object contains properties that are not defined in the XML, they remain unchanged. This attribute applies to each sub-block within the block that specifies it, unless the sub-block overrides the attribute value. For example, if the value in the `[GenericCreator]` element is **1**, then all XML blocks are skipped if the objects they define already exist in the PCD, unless `createMode` in a specific XML block is 2 or 3.
`version`	No	Specifies the version or a short description of the XML document. This attribute has no effect on the portal or the XML import.
`default.locale`	Yes	Specifies the default locale for an object if its `[Context]` element does not specify a locale attribute (`originalLocale`). Only meaningful if a `[Context]` element specifies an attribute of type `text`, for example, `Title`.

Example

```
<GenericCreator author="Joe Soap"
    version="Initial Content Bank 9/3/2005 6:19PM"
    mode="clean, execute" report.level="success" ignore="false"
    default.locale="en" createMode="2">
```

3.1.1 XML Element: [Property]

The `Property` element enables you to define global variables, and re-use them as needed anywhere in the XML document. This is useful for frequent occurrences of parameters in the XML document, such as the namespace and default locale.

Definition

Each `Property` element defines a single property name-value pair. A `Property` element must be nested within the root `[GenericCreator]` element. The `Property` element is defined as follows:

```
<Property name="<property_name>" value="<property_value>"/>
```

The following table describes the [`Property`] element attributes:

Attribute	Mandatory	Description
`name`	Yes	Specifies the name of the property variable.
`value`	Yes	Specifies the value of the property variable.

Usage

Any property name-value pair defined and nested in the `[GenericCreator]` element can be used elsewhere in the XML document as follows:

```
${<property_name>}
```

Example

The following example shows how to define global variables and use them within other elements:

```
<!—PROPERTY DEFINITION -->
<Property name="namespace" value="com.sap.portal"/>
<Property name="locale" value="en"/>
...
...
<!—PROPERTY USAGE -->
<Context name="${namespace}.urliview"
    template="par:/applications/com.sap.portal.urliviews/"
    objectClass="com.sapportals.portal.iview" create_as="0"
    domain="EP" originalLocale="${locale}" title="URL iView"/>
```

3.1.2 XML Element: [Context]

The [Context] element defines a semantic object to be created, deleted or updated in the PCD.

Definition

The [Context] element defines a semantic object. Certain attributes in the [GenericCreator] and [Context] elements determine which type of action is performed on the object: create, delete or update.

The [Context] element is defined as follows:

```
<Context name="<object_ID>" objectClass="<object_class>"
    template="<source_object>" create_as="<type_of_object" >
```

> **Important:** Typically, the [Context] element can support any attribute and sub-element, assuming it can be parsed by the object's handler and is valid for the object type. Some attributes and sub-elements are mandatory. This document describes only the mandatory, basic and commonly used attributes and sub-elements.

The following table describes the basic and commonly-used [Context] element attributes:

Attribute	Mandatory	Description
name	Yes	Specifies the object ID (technical name) of the object. Do not specify the full PCD path of the object. An object's PCD location is derived by its ID and the IDs of the objects that contain it (that is, [Context] elements in which it is nested). This is why nesting multiple [Context] elements is important for generating a hierarchy in the PCD.
title	No	Specifies the friendly name of an object. If this attribute is not defined, then the value of the name attribute is displayed in the Portal Catalog instead.

`parent`	No	Specifies the ID and path of the parent folder for the current object.
		
		Note: This attribute can only be used in a root `[Context]` element (one that is not nested in another `[Context]` element).
		
		The attribute enables you to associate the object to an existing PCD hierarchy, while defining it in the XML as a root `[Context]` tag area. This attribute is also useful for making specific modifications to a particular object located within a complex hierarchy.
		
		Important: If you want your content to be created in the standard *Portal Content* (`portal_content`) root folder of the Portal Catalog, you nevertheless need to define this in your XML. It is recommended that you create a parent `[Context]` element which nests your entire content script.
		
		For example:
		
		`<Context name="portal_content" objectClass="com.sap.portal.pcd .gl.GlContext" title="Portal Content">`
		
		`...`
		
		`</Context>`
		
		See also *Creating Hierarchies without Nested Elements* on page 34.

`objectClass`	Yes	Specifies the type of object. The following are valid values: • **`com.sap.portal.pcd.gl.GlContext`** (folders in the Portal Catalog) • **`com.sapportals.portal.iview`** (iViews) • **`com.sapportals.portal.layout`** (page layouts) • **`com.sapportals.portal.page`** (pages) • **`com.sapportals.portal.workset`** (worksets) • **`com.sapportals.portal.role`** (roles) • **`com.sapportals.portal.rolefolder`** (role folders) • **`com.sapportals.portal.system`** (systems) • **`com.sapportals.portal.transport .TransportPackage`** (packages) • **`com.sapportals.portal.desktop`** (portal desktops)

`template`	Yes[1]	Specifies one of the following: • The source object to which the current object is related through a delta link or copy. Use the following syntax: **`pcd:/<PCD_path>`** **Note:** It is possible to create a delta link to a source object that does not yet exist in the PCD. Technically, you will be generating a dangling link; however you can later create the missing object. • The portal component on which an object is based. Use the following syntax: **`par:/applications/<PAR_name>/components/<component_name>`** Only iViews, pages, page layouts and systems can be based on portal components. **Note:** The portal components on which objects are based must be deployed to the portal before importing an XML script. This attribute does not specify if the object is an object template. To make an object a template, use the `IsTemplate` property (or use the `[Attribute]` and `[AttributeValue]` elements). Valid values are **`true`** and **`false`**.

`create_as`	Yes	Specifies the relationship of the object to the template or portal component on which it is based. This attribute is dependent on the `template` attribute. The following are valid values: • **`0`**: For the following cases: ○ To create a new object that is based directly on a portal component. The `template` attribute specifies the portal component. ○ To make a copy of an object that already exists in the PCD. The new object and the copied object become siblings and share the same source object (via a delta link) or portal component. The `template` attribute specifies the source object. • **`1`**: To create an object that is a delta link of an object that already exists in the PCD. The `template` attribute specifies the source object. **Important:** This value cannot be used if the `template` attribute specifies a portal component. It must specify a semantic object. Typically, delta link objects inherit properties and values from their source objects. To assign different object properties, use the `[Attribute]` and `[AttributeValue]` elements. For code samples, see *Creating iViews* on page 20.
`originalLocale`	Yes[2]	Specifies the original locale of the object. **Important:** Note that this value must only be set for standalone or unit objects in the PCD. A standalone or unit object is one that is not currently assigned to another object type. For example, an iView in a page, a workset in a role, or a page in a role must not be assigned this value.
`PrimaryLayout`	Yes[3]	Specifies which page layout assigned to a page is the primary layout. Typically, a page can only have one primary page layout. If you assign more than one primary layout, the last one assigned is the primary layout. The following are valid values: • **`true`**: The page layout is the primary layout. • **`false`**: The page layout is not the primary layout.
`container`	Yes[3]	Specifies the container name in a page layout in which to position the iView/page. The container name must exist in the primary page layout defined for the page to which the current object is assigned.

`Domain`	No	Specifies the domain of the object. This attribute is for SAP internal content developers only, in order to support in-house translation. Do not use **EP**; it is reserved for SAP content.
`Collection`	No	Specifies the collection setting of the object. This attribute is for SAP internal content developers only, in order to support in-house translation. Do not use **IP_PTL_INITIAL_CONTENT**; it is reserved for SAP content.
`ignore`	No	Functions in the same way as the `ignore` attribute in the root `[GenericCreator]` element. For more information, see *XML Element: [GenericCreator]* on page 6. If you define a value for this attribute in the `[Context]` element, it overrides the value defined in the `[GenericCreator]` element. However, if no value is defined in the `[Context]` element, the value defined in the `[GenericCreator]` element is used.
`createMode`	No	Functions in the same way as the `createMode` attribute in the root `[GenericCreator]` element. For more information, see *XML Element: [GenericCreator]* on page 6. If you define a value for this attribute in the `[Context]` element, it overrides the value defined in the `[GenericCreator]` element. However, if no value is defined in the `[Context]` element, the value defined in the `[GenericCreator]` element is used.

[1] Mandatory only for delta link target objects and objects that link directly to a portal component.

[2] To be used ONLY for standalone or unit objects in the PCD. For example, do not apply this attribute to an iView inside a page, or a workset inside a role.

[3] Mandatory for portal pages and iViews only that are positioned inside a portal page.

Usage

To place an object inside another object – for example, an iView in a page – nest the `[Context]` element of the child object within the `[Context]` element of the parent object. You can also use the `parent` attribute in the child `[Context]` element instead of nesting elements.

Keep in mind that the Portal Catalog displays only folders and standalone or unit objects (parent objects). To access nested child objects in the portal, you need to open the parent object in its editor.

Example

```
<Context parent="portal_content" name="myFolder"
    objectClass="com.sap.portal.pcd.gl.GlContext"
    title="My Folder" originalLocale="en">
```

3.1.3 XML Element: [Action]

XML elements of type [Action] differ in concept and syntax to XML elements of type [Context]. The [Action] element generally performs an action within the portal, instead of generating or updating a semantic object in the PCD.

Actions are typically general; they tend not to be specific to a particular object type (although it is possible to develop a handler of type [Action] that operates on a particular content type).

Definition

The [Action] element is defined as follows:

```
<Action id="<handler_name>" ignore="<mode>" />
```

The following table describes the [Action] element attributes:

Attribute	Mandatory	Description
id	Yes	Specifies the action.
ignore	No	Functions in the same way as the ignore attribute in the root [GenericCreator] element. For more information, see *XML Element: [GenericCreator]* on page 6. If you define a value for this attribute in the [Action] element, it overrides the value defined in the [GenericCreator] element. However, if no value is defined in the [Action] element, the value defined in the [GenericCreator] element is used.

There may be additional attributes specific to each handler.

Usage

- [Action] elements cannot be nested within each other, nor can they be nested within [Context] elements, or vice versa.
- [Action] elements are only executed when the script is parsed in execute mode (the mode is specified in the mode attribute specified in the [GenericCreator] root element).

3.1.4 XML Element: [Attributes], [Attribute] and [AttributeValue]

The [Attributes], [Attribute] and [AttributeValue] elements enable you to define properties (metadata) for semantic objects in the PCD. In the portal, properties are viewed and edited within the Property Editor.

Typically, some object properties and values are defined in portal components (in PAR files). Therefore, use [Attributes], [Attribute] and [AttributeValue] elements in the following cases:

- To assign a different value to an existing property so that it does not inherit the predefined value from a source object (in the case of a delta link) or its portal component.
- To assign values to existing properties that are not initially assigned a value.
- To define new properties

The [Attributes], [Attribute] and [AttributeValue] elements can also be used to pass information to configure an action.

Definition

The [Attribute] element defines a property and the [AttributeValue] elements nested within an [Attribute] element define the values for that property.

All [Attribute] elements for a [Context] or [Action] element must be nested within an [Attributes] element, which takes no attributes.

The format for [Attribute] and [AttributeValue] elements is as follows:

```
<Attributes>
    <Attribute name="<attribute_name>" type="<attribute_type>"
        Inheritance="<attribute_inheritance>">
        <AttributeValue value="<value>" locale="<locale>" />
    </Attribute>
    ... (other attributes)
</Attributes>
```

The following describes the [Attribute] element attributes:

Attribute	Mandatory	Description and Valid Values
name	Yes	Specifies the name of the property. Property names can be found by opening the property editor for an object in the Portal Catalog. Also, the portal API provides interfaces that define constants for property names of semantic objects, for example, IAttriView for properties of iViews.

`type`	Yes	Specifies the property's data type. The following are valid values: • **`string`** • **`text`** • **`integer`** • **`boolean`** • **`double`** • **`long`**
`Inheritance`	Yes	Specifies the inheritance mode of the property. This property is currently not supported by the portal. Set this to **`NONFINAL`**.

The following describes the `[AttributeValue]` element attributes:

Attribute	Mandatory	Description and Valid Values
`value`	Yes	Specifies the value of the property variable.
`locale`	Yes[1]	Specifies the locale of the property's value (for text-based data, where `type="text"`).

[1] Mandatory only for properties of type `text`.

Example

```
<Context>
    ...
    <Attribute name="com.sap.portal.pcm.Description" type="text">
        <AttributeValue value="Schedule Processing" locale="en" />
    </Attribute>
    ...
</Context>
```

3.1.4.1 Meta-Attributes

To define meta-attributes, nest `[Attribute]` elements. The following defines the `category` meta-attribute for the `myProperty` attribute:

```
<Context>
    ...
    <Attribute name="myProperty" type="text">
        <AttributeValue value="ABC" locale="en" />
        <Attribute name="category" type="text">
            <AttributeValue value="myProperties" locale="en"/>
        </Attribute>
    </Attribute>
    ...
</Context>
```

4 Available Semantic Objects and Actions

This section describes what you can do by importing an XML file.

Semantic Objects

The following lists the semantic objects that can be created or modified via an imported XML file, and the object class for specifying the object type in a `[Context]` element:

Semantic Object	Object Class
Folders (in Portal Catalog)	**`com.sap.portal.pcd.gl.GlContext`**
iViews	**`com.sapportals.portal.iview`**
Pages	**`com.sapportals.portal.page`**
Page Layouts	**`com.sapportals.portal.layout`**
Worksets	**`com.sapportals.portal.workset`**
Roles	**`com.sapportals.portal.role`**
Role Folders	**`com.sapportals.portal.rolefolder`**
Systems	**`com.sapportals.portal.system`**
Translation Worklists	**`com.sap.portal.pcd.translation.TranslationWorklist`**
Desktops	**`com.sapportals.portal.desktop`**
Transport Packages	**`com.sapportals.portal.transport.TransportPackage`**

Semantic objects are created and modified with the `[Context]` element. To specify the type of semantic object, set the `objectClass` attribute to the appropriate value shown in the table above.

In addition to creating semantic objects and setting attributes, you can also perform the following tasks with the `[Context]` element:

- **Assign iViews to a Page:** You can also assign iViews and pages to worksets, or worksets to roles.
- **Create Related Items:** You can create Related Items links or Dynamic Navigation iViews for a specific iView or page.

Actions

The following are actions that can be executed via an imported XML file, and the ID for specifying the action with the `[Action]` element:

Action	ID
Deleting Content	**`com.sap.portal.gc.deepCleaner`**
Adding/Removing System Aliases	**`com.sap.portal.alias.handler`**
Running Another Script	**`com.sap.portal.script.runner`**
Setting Permissions	This tag has a special syntax, and does not use the `<Action>` tag.

Actions are executed with the `[Action]` element. To specify an action, set the `id` attribute to the ID for the specific action in the table above.

4.1 Code Samples for Semantic Objects

This section contains basic XML code samples for creating and modifying semantic objects.

4.1.1 Creating Folders in the Portal Catalog

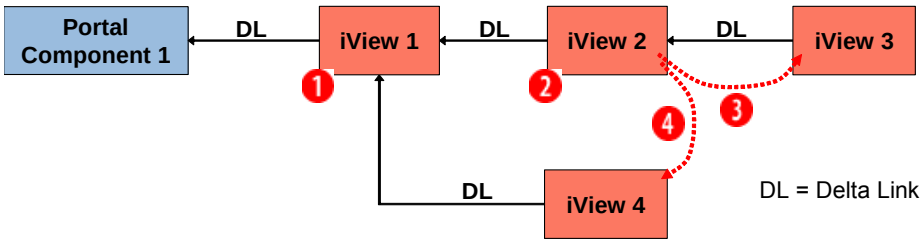
The following creates a folder named `Migrated Content` in the root `Portal Content` folder.

```
<Context name="portal_content"
objectClass="com.sap.portal.pcd.gl.GlContext"
title="Portal Content">
    <Context name="com.sap.portal.migrated"
    objectClass="com.sap.portal.pcd.gl.GlContext"
    title="Migrated Content"/>
</Context>
```

4.1.2 Creating iViews

iViews, pages and systems can be created directly from portal components or as copies or delta links of other portal objects. The method for creating iViews, pages and systems can be specified in the XML, generally with the attributes `template` and `create_as`.

The following shows several ways to create iViews with varying dependencies. In this example, iView 1, 2, 3, and 4 are all based on the same portal component. The legend describes the `template` and `create_as` attributes for defining each iView.



1. template="par:/applications/portalcomponent1"; create_as="0"
2. template="pcd:/portal_content/myFolder/iView1"; create_as="1"
3. template="pcd:/portal_content/myFolder/iView2"; create_as="1"
4. template="pcd:/portal_content/myFolder/iView2"; create_as="0"

The following examples show how to create an iView based on a portal component, or based on a copy or delta link of another portal object.

4.1.2.1 Based on a Portal Component

The following creates an iView from a PAR file and based on the portal component `com.sap.portal.ivs.alias_editor.AliasEditor`.

```
<Context name="${namespace}.aliasEditor"
title="System Alias Editor"
template="par:/applications/com.sap.portal.ivs.alias_editor/compone
nts/AliasEditor" objectClass="com.sapportals.portal.iview"
create_as="0" collection="${collection}" domain="EP"
originalLocale="${locale}">
    <Attributes>
        <Attribute
        name="com.sap.portal.reserved.iview.IsolationMode"
        type="string">
            <AttributeValue value="PUMPED"/>
        </Attribute>
        <Attribute name="com.sap.portal.iview.HeightType"
        type="string">
            <AttributeValue value="FULL_PAGE"/>
        </Attribute>
        <Attribute name="com.sap.portal.iview.ShowTray"
        type="string">
            <AttributeValue value="false"/>
        </Attribute>
        <Attribute name="com.sap.portal.reserved.iview.ParamList"
        type="string">
            <AttributeValue value="*"/>
        </Attribute>
    </Attributes>
</Context>
```

See iView 1 in the figure above.

4.1.2.2 Based on iView (Delta Link)

The following creates an iView based on a delta link from the iView located at `portal_content/com.sap.pct/admin.templates/iviews/${namespace}.contentCatalog`.

```
<Context name="${namespace}.contentCatalog"
template="portal_content/com.sap.pct/admin.templates/iviews/${names
pace}.contentCatalog" objectClass="com.sapportals.portal.iview"
create_as="1" container="com.sap.portal.reserved.layout.Cont2">
    <Attributes>
        <Attribute
        name="com.sap.portal.reserved.iview.IsolationMode"
        type="string">
            <AttributeValue value="URL"/>
        </Attribute>
        <Attribute name="com.sap.portal.iview.HeightType"
        type="string">
            <AttributeValue value="FULL_PAGE"/>
        </Attribute>
        <Attribute name="com.sap.portal.iview.ShowTray"
        type="string">
            <AttributeValue value="false"/>
        </Attribute>
    </Attributes>
</Context>
```

See iView 2 in the figure above.

4.1.2.3 Based on iView (Copy)

The following creates an iView by copying the iView located at `portal_content/com.sap.pct/admin.templates/iviews/${namespace}.contentCatalog`.

```
<Context name="${namespace}.contentCatalog"
template="portal_content/com.sap.pct/admin.templates/iviews/${names
pace}.contentCatalog" objectClass="com.sapportals.portal.iview"
create_as="0" container="com.sap.portal.reserved.layout.Cont2">
    <Attributes>
        <Attribute
        name="com.sap.portal.reserved.iview.IsolationMode"
        type="string">
            <AttributeValue value="URL"/>
        </Attribute>
        <Attribute name="com.sap.portal.iview.HeightType"
        type="string">
              <AttributeValue value="FULL_PAGE"/>
        </Attribute>
        <Attribute name="com.sap.portal.iview.ShowTray"
        type="string">
            <AttributeValue value="false"/>
        </Attribute>
    </Attributes>
</Context>
```

See iView 3 in the figure above.

4.1.3 Creating Pages

The following creates a page called `Portal Information`.

```
<Context name="${namespace}.portal_information"
template="portal_content/com.sap.pct/admin.templates/pages/${namesp
ace}.portalpagetemplate" objectClass="com.sapportals.portal.page"
create_as="1" title="Portal Information"/>
```

4.1.3.1 Assigning iViews to a Portal Page

The following creates a page called `User Data Import`.

```
<Context name="${namespace}.batchUpload"
template="portal_content/com.sap.pct/admin.templates/pages/${namesp
ace}.portalpagetemplate" objectClass="com.sapportals.portal.page"
create_as="1" title="User Data Import">

    <Context name="${namespace}.fullWidth"
    template="portal_content/com.sap.pct/admin.templates/layouts/
    ${namespace}.fullWidth"
    objectClass="com.sapportals.portal.layout" create_as="1"
    PrimaryLayout="true">
    </Context>

    <Context name="${namespace}.batchUpload"
    template="portal_content/com.sap.pct/admin.templates/iviews/
    ${namespace}.batchUpload"
    objectClass="com.sapportals.portal.iview" create_as="1"
    title="User Data Import">
        <Attributes>
            <Attribute name="com.sap.portal.iview.ShowTray"
            type="string">
                <AttributeValue value="false"/>
            </Attribute>
        </Attributes>
    </Context>

    <Attributes>
        <Attribute name="com.sap.portal.iview.TrayType"
        type="string">
            <AttributeValue value="TRANSPARENT"/>
        </Attribute>
        <Attribute
        name="com.sap.portal.reserved.iview.IsolationMode"
        type="string">
            <AttributeValue value="URL"/>
        </Attribute>
    </Attributes>

</Context>
```

4.1.3.2 Creating Related Items

The following creates an iView with the ID `DNiView`, and then creates for this iView a Dynamic Navigation iView based on a delta link of the page at the PCD address `portal_content/myIviews/pages/myPage`.

```
<Context name="DNiView" objectClass="com.sapportals.portal.iview"
template="portal_content/com.sap.pct/admin.templates/iviews/com.sap
.portal.pageBuilderDefault" create_as="1" >

    <Context name="DN1" objectClass="com.sapportals.portal.page"
    create_as="1" relatedItem="true"
    relatedItemType="dynamicNavigation"
    template="portal_content/myIviews/pages/myPage"/>

</Context>
```

The `relatedItemType` attribute can be set to the following values:

- **`dynamicNavigation:`** Creates a Dynamic Navigation iView.
- **`relatedLinks:`** Creates a Related Item link.
- **`targetComponents:`** Creates a Drag&Relate link.

4.1.4 Creating Page Layouts

The following creates a new page layout called `fullWidth` and assigns it to the page `Portal Information`.

```
<Context name="${namespace}.fullWidth"
template="par:/applications/com.sap.portal.layouts.default/componen
ts/fullWidth" objectClass="com.sapportals.portal.layout"
create_as="0" collection="${collection}" domain="EP"
originalLocale="${locale}"/>

<Context name="${namespace}.portal_information"
template="portal_content/com.sap.pct/admin.templates/pages/${namesp
ace}.portalpagetemplate" objectClass="com.sapportals.portal.page"
create_as="1" title="Portal Information">

    <Context name="${namespace}.fullWidth"
    template="portal_content/com.sap.pct/admin.templates/
    layouts/${namespace}.fullWidth"
    objectClass="com.sapportals.portal.layout" create_as="1"
    PrimaryLayout="true"/>
    </Context>

</Context>
```

4.1.5 Creating Worksets

The following creates a workset called `Company`.

```
<Context name="${namespace}.home.company"
objectClass="com.sapportals.portal.workset" entryPoint="false"
asUnit="true" title="Company" collection="${collection}"
domain="EP" originalLocale="${locale}">

    <Attributes>
        <Attribute name="com.sap.portal.navigation.MergeId"
        type="string">
            <AttributeValue value="${namespace}.home.company"/>
        </Attribute>
    </Attributes>

</Context>
```

4.1.6 Creating Roles

The following creates a role called `Delegated User Admin`.

```
<Context name="${namespace}.delegated_user_admin_role"
objectClass="com.sapportals.portal.role" entryPoint="false"
collection="${collection}" domain="EP" originalLocale="${locale}"
title="Delegated User Admin">
    <Attributes>
        <Attribute name="UME.Delegated_Admin" type="string">
            <AttributeValue value="true"/>
        </Attribute>
    </Attributes>
</Context>
```

4.1.7 Creating Role Folders

The following creates a role folder.

```
<Context name="portal"
objectClass="com.sapportals.portal.rolefolder" entryPoint="false"
title="Portal"/>
```

4.1.8 Creating Systems

The following creates a JDBC system.

```
<Context name="${namespace}.JDBCConnectorSystem"
template="par:/applications/com.sap.portal.systems.jdbc/components/
jdbc_system" objectClass="com.sapportals.portal.system"
create_as="0" collection="${collection}" domain="EP"
originalLocale="${locale}"/>
```

4.1.9 Creating Translation Worklists

The following creates a translation worklist named `Sample Translation Worklist`. The worklist is made up of content from the `portal_content/gc_samples/content_iviews` and `portal_content/gc_samples/systems` folders, as defined in the `root` attribute.

The `filter` attribute enables you to supply a JNDI search string to specify a subset of objects in the root folders.

```
<Context name="sample_translation_wl"
objectClass="com.sap.portal.pcd.translation.TranslationWorklist"
collection="${collection}" domain="domain"
originalLocale="${locale}"
root="pcd:portal_content/gc_samples/content_iviews,
pcd:portal_content/gc_samples/systems"
filter="(|(com.sap.portal.pcd.gl.AtomicName=*)(com.sap.portal.pcd.g
l.ObjectClass=com.sap.portal.pcd.gl.GlContext))"
title="Sample Translation Worklist">

    <Attributes>
        <Attribute name="com.sap.portal.pcm.Description"
        type="string">
            <AttributeValue value="Translation Worklist"/>
        </Attribute>
    </Attributes>

</Context>
```

4.1.10 Creating Desktops

The following creates a desktop named `Default Portal Desktop`.

The list of themes – in the attribute – are written in StringList format and are composed of two strings: the name of the theme and the PCD location of the theme.

```
<Context name="${namespace}.defaultDesktop"
objectClass="com.sapportals.portal.desktop" create_as="0"
asUnit="true" collection="${collection}" domain="EP"
originalLocale="${locale}" defaultTheme="sap_standard"
defaultFwPage="${namespace}.frameworkpage" title="Default Portal
Desktop">
    <Attributes>
        <Attribute
        name="com.sappportals.portal.desktop.defaultFwPage"
        type="string">
            <AttributeValue value="${namespace}.frameworkpage"/>
        </Attribute>
        <Attribute
        name="com.sappportals.portal.desktop.defaultTheme"
        type="string">
            <AttributeValue value="sap_tradeshow"/>
        </Attribute>
        <Attribute name="com.sappportals.portal.desktop.allThemes"
        type="string">
            <AttributeValue value="13:sap_tradeshow39:pcd:portal_co
ntent/themes/sap_tradeshow"/>
            <AttributeValue value="12:sap_standard038:pcd:portal_co
ntent/themes/sap_standard"/>
            <AttributeValue value="10:sap_chrome00036:pcd:portal_co
ntent/themes/sap_chrome"/>
            <AttributeValue value="07:sap_hcb00000033:pcd:portal_co
ntent/themes/sap_hcb"/>
            <AttributeValue value="12:sap_highcont038:pcd:portal_co
ntent/themes/sap_highcont"/>
        </Attribute>
    </Attributes>
    <Context name="frameworkPages"
    objectClass="com.sap.portal.pcd.gl.GlContext">
        <Context name="${namespace}.frameworkpage"
        template="portal_content/com.sap.pct/every_user/general
        /${namespace}.frameworkpage" create_as="1"
        objectClass="com.sapportals.portal.page"/>
    </Context>
</Context>
```

4.1.11 Creating Transport Packages

The following creates a transport package named `Sample Content Package`.

```
<Context parent="pcd:portal_content/package_tests"
name="sample_package"
objectClass="com.sapportals.portal.transport.TransportPackage"
collection="${collection}" domain="EP11" originalLocale="${locale}"
root="pcd:portal_content/folders"
filter="(|(com.sap.portal.pcd.gl.AtomicName=*)(com.sap.portal.pcd.
gl.ObjectClass=com.sap.portal.pcd.gl.GlContext))"
title="Sample Content Package">

    <Attributes>
        <Attribute name="com.sap.portal.pcm.Description"
        type="string">
            <AttributeValue value="Transport Package"/>
        </Attribute>
    </Attributes>

</Context>
```

4.2 Code Samples for Actions

This section contains basic XML code samples for executing actions.

Each heading in this section contains the ID suffix for each action, which should be preceded by `com.sap.portal`. For example, the action ID for deleting content is `com.sap.portal.gc.deepCleaner`.

4.2.1 Deleting Content: [gc.deepCleaner]

This action enables you to delete content in the PCD. You specify the start folder and the action is performed recursively. You can also specify content to exclude from the deletion.

This action is different to the `clean` mode execution specified in `[Context]` elements. Whereas the `clean` mode only deals with objects specified in the XML, the deep clean action is performed on any semantic object located in the specified folder.

> **Warning:** This handler is still in development. It is advised to use this action with extreme caution. In some instances the `[DeepCleaner]` may unknowingly delete PCD data that is not within the specified folder. This issue is under investigation.

In addition to the basic attributes required by the `[Action]` element, the following attributes are expected by this action:

Attribute	Mandatory	Description
`id`	Yes	**`com.sap.portal.gc.deepCleaner`**
`root.folder`	Yes	The ID of the folder from which to start the deep clean process
`exclude.folder`	No	The ID of the folder which the deep cleaner must ignore. This must be a folder within the hierarchy of the `root.folder` attribute. You must enter an absolute path; in other words, do not enter an ID that is relative to the `root.folder` attribute. You cannot enter more than one folder to exclude. The exclusion is recursive from the specified folder onward.

The following deletes all content in the `portal_content/test` folder except for the content in the `portal_content/test/myFolder` folder.

```
<Action id="com.sap.portal.gc.deepCleaner" ignore="false"
root.folder="pcd:portal_content/test"
exclude.folder="pcd:portal_content/test/myFolder" />
```

4.2.2 Adding/Removing System Aliases: [alias.handler]

This action enables you to add and remove a system alias, and to set a specific system alias as the system's default alias.

In addition to the basic attributes required by the `[Action]` element, the following attributes are expected by this action:

Attribute	Mandatory	Description
`id`	Yes	**`com.sap.portal.alias.handler`**
`system`	Yes	The ID of the system to which to modify its aliases

The following adds aliases `a1`, `a2`, `a3` and `a4`, sets the default alias to `a3`, and deletes aliases `d1` and `d2` for the system whose PCD address is `portal_content/samples/mySystem`.

```
<Action id="com.sap.portal.alias.handler"
system="portal_content/samples/mySystem">
    <Attributes>
        <Attribute name="addAlias">
            <AttributeValue value="a1"/>
            <AttributeValue value="a2"/>
            <AttributeValue value="a3"/>
            <AttributeValue value="a4"/>
        </Attribute>
        <Attribute name="changeDefaultAlias">
            <AttributeValue value="a3"/>
        </Attribute>
        <Attribute name="removeAlias">
            <AttributeValue value="d1"/>
            <AttributeValue value="d2"/>
        </Attribute>
     </Attributes>
</Action>
```

4.2.3 Running Another Script: [script.runner]

This action enables you to run another XML script from within a XML script. You can run the script several times, in a loop, and set generic properties to the script.

In addition to the basic attributes required by the `[Action]` element, the following attributes are expected by this action:

Attribute	Mandatory	Description
`id`	Yes	**`com.sap.portal.script.runner`**
`file.name`	Yes	The full path of the script to run. The script must be located on the portal server.
`loop`	Yes	The number of times to run the script.

You can set general properties for all iterations of the script by supplying an attribute named `external.properties`.

You can also set general properties so that the property has a different value for each iteration of the script. You can define these properties in an attribute named `loop.properties`. For each property defined within this attribute, supply several values, one for each iteration of the script. In other words, the number of values for each property should be equal to the value defined for the `loop` attribute in the `[action]` tag.

The following sample starts running the script whose path is `c:\usr\myScript.xml`. The script is run twice, with the property `myProperty1` set to `25` on the first iteration and `50` on the second iteration, and the property `myProperty2` set to `10` on the first iteration and `20` on the second iteration.

For both iterations, the property `myGeneralProperty1` is set to `Mike` and `myGeneralProperty2` is set to `Joe`.

```
<Action id="com.sap.portal.script.runner"
file.name="c:\usr\myScript.xml" loop="2">
    <Attributes>
        <Attribute name="external.properties">
            <Attribute name="myGeneralProperty1">
                <AttributeValue value="Mike"/>
            </Attribute>
            <Attribute name="myGeneralProperty2">
                <AttributeValue value="Joe"/>
            </Attribute>
        </Attribute>
        <Attribute name="loop.properties">
            <Attribute name="myProperty1">
                <AttributeValue value="25" />
                <AttributeValue value="50"/>
            </Attribute>
            <Attribute name=" myProperty2">
                <AttributeValue value="10"/>
                <AttributeValue value="20"/>
            </Attribute>
        </Attribute>
    </Attributes>
</Action>
```

4.2.4 Setting Permissions

This action enables you to set the permissions for any portal object by replacing the ACL (access control list) for the portal object. The ACL is a collection of ACEs (access control entries), which define specific permissions for specific users, groups or roles.

For more information on portal permissions, see the documentation on the SAP Help Portal (help.sap.com) at *Documentation → SAP NetWeaver → People Integration → Portal → Administration Guide → System Administration → Permissions*, *Role*/*User Distribution*, *and Object Locking → Portal Permissions*.

> **Note:** This action does not use the `[Action]` element and has its own element and syntax.

To set permissions, use the following elements:

- **[ACL]:** Create an ACL element for each portal object whose permissions you want to set. The element takes the following attributes:

Attribute	Mandatory	Description
`objectID`	Yes	The PCD address of the portal object whose permissions you want to set
`handlerId`	Yes	Always set to `ACL`

  Within each `[ACL]` element, nest an `[ACEs]` element.

- **[ACEs]:** Nest an `[ACEs]` element inside an `[ACL]` element. Within the `[ACEs]` element, nest one or more `[ACE]` elements for the current portal object.

- **[ACE]:** Nest one or more `[ACE]` elements inside an `[ACEs]` element for each ACE that you want to create for the current portal object. The element takes the following attributes:

Attribute	Mandatory	Description
`type`	Yes	The type of principle specified by the `principalID` attribute, either `user`, `group` or `role`
`principalID`	Yes	The principle receiving the permission
`permission`	Yes	The permission to grant to the principle specified by the `principalID` attribute The following are valid values: • **NONE:** No administration permission. • **Pcd.Read:** The principle can read the object. • **Pcd.ReadWrite:** The principle can read and change the object. • **Pcd.FullControl:** The principle can read, change and delete the object. • **Owner:** The principle can read, change and delete the object, and change the permissions of the object.
`endUserRead`	No	Indicates whether the principle gets end user permission. Valid values are `true` or `false` (default).

`roleAssign`	No	Indicates whether the principle gets role assigner permission. Valid values are `true` or `false` (default). This permission can only be assigned for role objects, and folders containing role objects that inherit permissions from the folder.

The following example assigns permissions to the portal object with the PCD address `pcd:portal_content`:

```
<ACL objectID="pcd:portal_content" handlerId="ACL">
    <ACEs>
        <ACE type="role"
            principalID="pcd:portal_content/administrator/
content_admin/content_admin_role"
            permission="Pcd.FullControl"
            endUserRead="true" />
        <ACE type="group"
            principalID="GRUP.SUPER_GROUPS_DATASOURCE.EVERYONE"
            permission="NONE"
            endUserRead="true"
            roleAssign="true" />
        <ACE type="role"
            principalID="pcd:portal_content/administrator
/super_admin/super_admin_role"
            permission="owner"
            endUserRead="true"
            roleAssign="true" />
        <ACE type="role"
            principalID="pcd:portal_content/administrator
/system_admin/system_admin_role"
            permission="Pcd.ReadWrite"
            endUserRead="true" />
    </ACEs>
</ACL>
```

Exporting Permissions

You can export to an XML file the existing permissions of objects in one portal, and then set those permissions in another portal by importing the XML file.

For more information on exporting permissions, see *Transporting Permissions* of the *Portal Permissions* section of the *Administration Guide*.

For more information on viewing all the permissions in the portal, see *Viewing Permission Structures in the Portal* in the *Using the Permission Editor* section of *Portal Permissions*.

4.3 Tips and Tricks

4.3.1 General Tips

- Avoid using special characters in the object ID of content objects.
- Use the correct data types and locale for properties in content objects.
- Angle brackets (< >) are reserved for XML script. If you need to use them elsewhere, use **`&lt`** and **`&gt`**.

4.3.2 Executing Specific XML Blocks

If you want to re-use an XML script to make pinpoint changes to content that has already been created using the file, apply the `ignore` attribute to skip all blocks in the XML document except for those you want to execute. The `ignore` attribute is applied to each block within the block that defines it, unless specified otherwise.

For example, configure the XML document as follows:

1. Set the `ignore` value to **`true`** in the `[GenericCreator]` root element.
2. For all blocks to be executed, insert the `ignore` attribute and set it to **`false`**.
3. In all child blocks that should be skipped, make sure they do not declare the `ignore` attribute. If they do, set the value to **`true`**.

This procedure supports both `execute` and `clean` modes.

4.3.3 Creating Hierarchies without Nested Elements

Typically, you nest one `[Context]` element within another to generate object hierarchies in the PCD. However, you can instead create hierarchies by using the `parent` attribute in the `[Context]` element. The attribute specifies the ID and path of the parent folder for the defined object.

> **Note:** The `parent` attribute can only be used in a root `[Context]` element, that is, you cannot use it in a `[Context]` element that is nested in another `[Context]` element.

The following shows the use of the `parent` attribute to create a nested iView:

```
<Context name="${namespace}.portletProxyIview"
parent="portal_content/com.sap.pct/templates/iviews" ignore="false"
template="par:/applications/com.sap.portal.ivs.wsrpservice/componen
ts/ProxyPortalComponent" objectClass="com.sapportals.portal.iview"
create_as="0" collection="${collection}" domain="EP"
originalLocale="${locale}" title="Portlet Proxy iView" />
```

5 Exporting/Importing Content and Actions

The portal provides the following XML Content and Actions tools:

- **Export Tool:** Enables you to create an XML file based on existing content, as described in Section 5.1, *Exporting Content*. This file can be edited and imported into a portal (via the import tool) in order to create content.
- **Import Tool:** Enables you to import an XML file in order to create content and perform actions, as described in Section 5.2, *Importing Content and Actions*.

By default, these tools are assigned to the standard system administration role, and can be accessed by navigating to *System Administration* → *Transport* → *XML Content and Actions*.

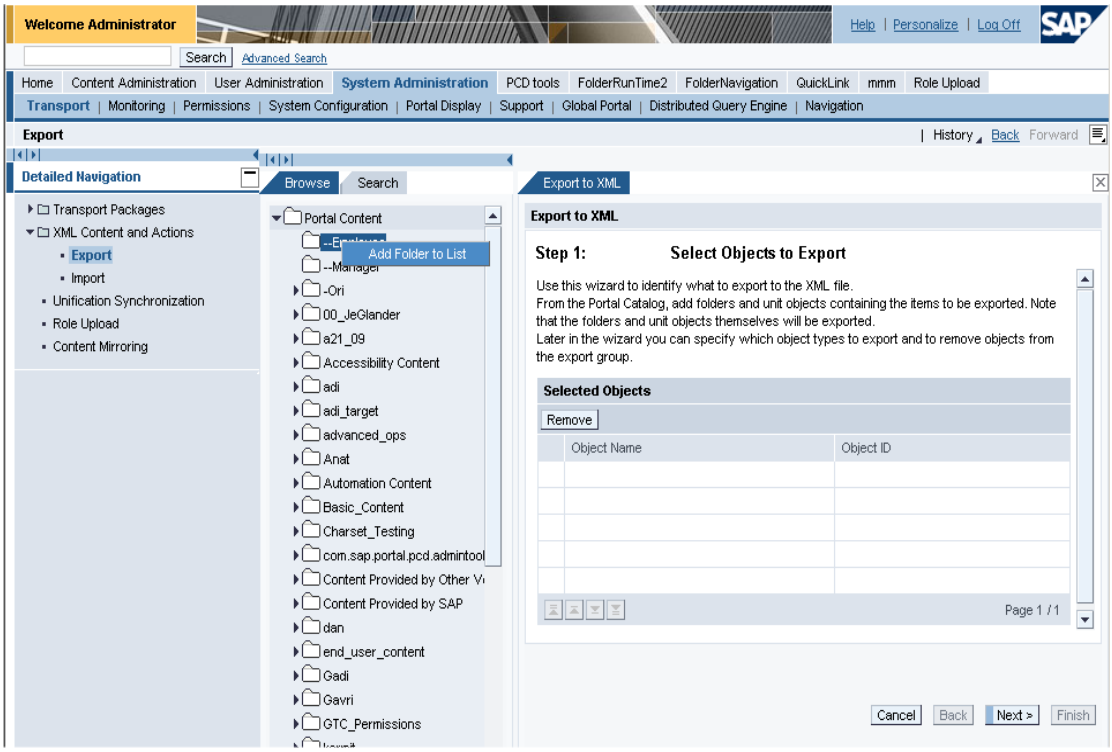
5.1 Exporting Content

The *XML Content and Actions* export tool automatically creates an XML file based on existing portal content. This file can be edited and imported into a portal in order to create content.

You can use this tool to re-create or move content from one area of the Portal Catalog to another.

Procedure

1. In the portal, navigate to *System Administration* → *Transport* → *XML Content and Actions* → *Export*.

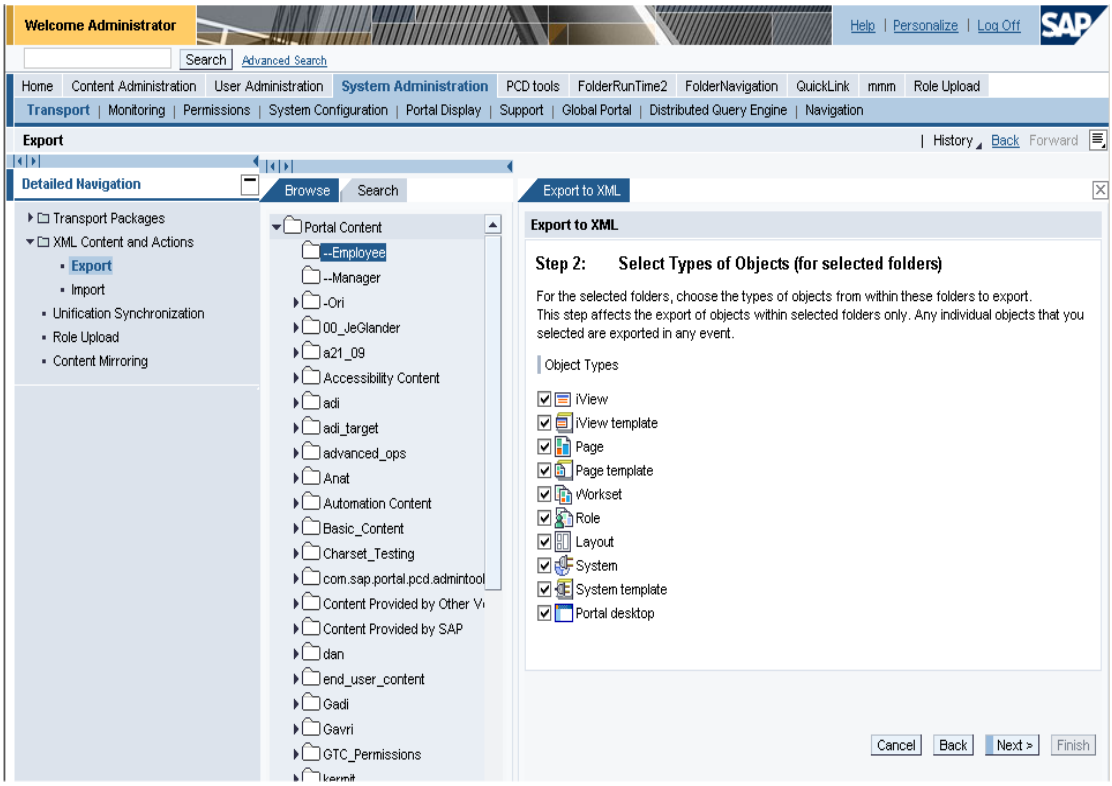


2. Select the content from which to generate the XML by right-clicking each object in the Portal Catalog and clicking *Add <Object> to List*.

   Click *Next*.

3. Select the type of objects to export.

   This step is only needed, and the pane is only displayed, if at least one folder was selected in the previous step. This pane determines the types of objects that are exported from within the selected folders.

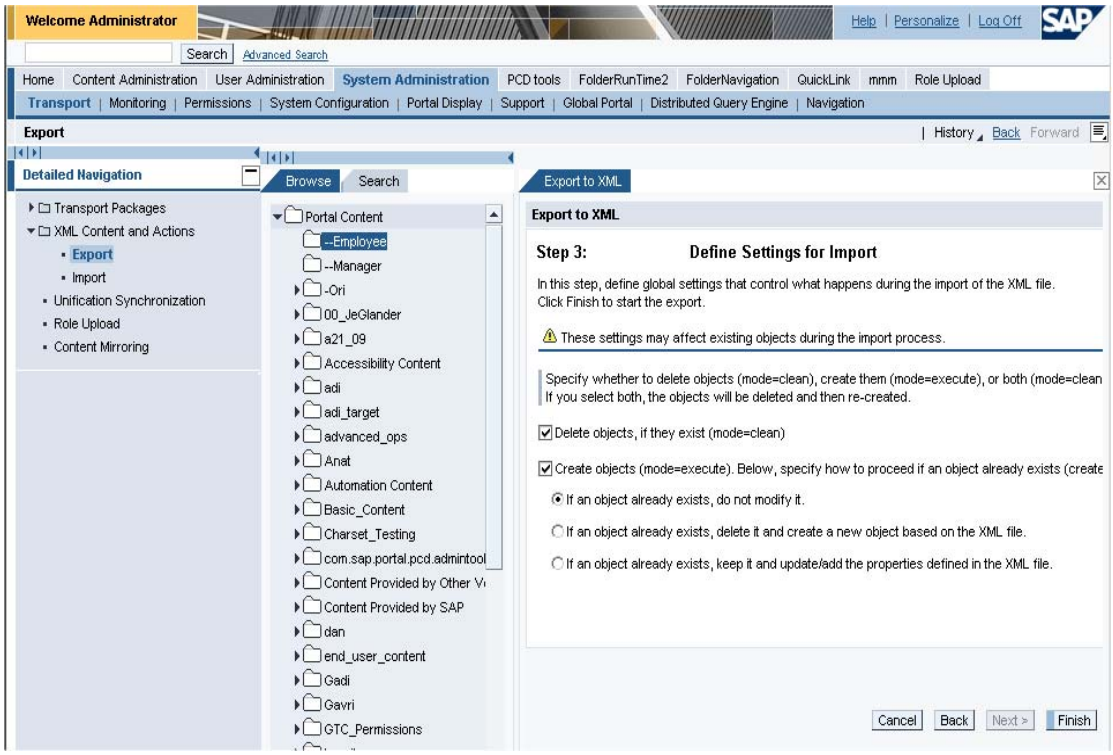


This step does not affect whether the tool exports the individual portal objects (that is, non-folder objects) that were selected in the previous step.

For example, if you select in this step only iViews, only iViews from within selected folder are exported. All the individual iViews, pages, roles and other objects that were selected in the previous step are exported in any event.

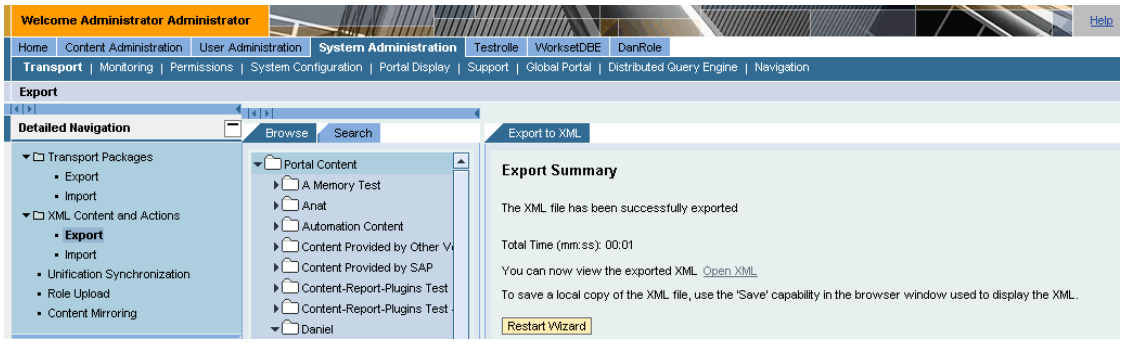
Click *Next*.

4. Select the settings to be used when the exported XML file is imported into a portal. This step affects XML attributes in the exported file.



Click *Finish*.

The XML file is created, and a link to the file (**Open XML**) is displayed. Click the link to view the file.



5.1.1 Location of XML Files

Copies of the XML files created with the export tool are placed in the folder `portalapps\com.sap.portal.content.export\xml` under the portal root folder.

These files can be large. System administrations should make sure to clean out this folder from time to time.

5.2 Importing Content and Actions

The *XML Content and Actions* import tool enables you to import an XML file for creating content and performing actions.

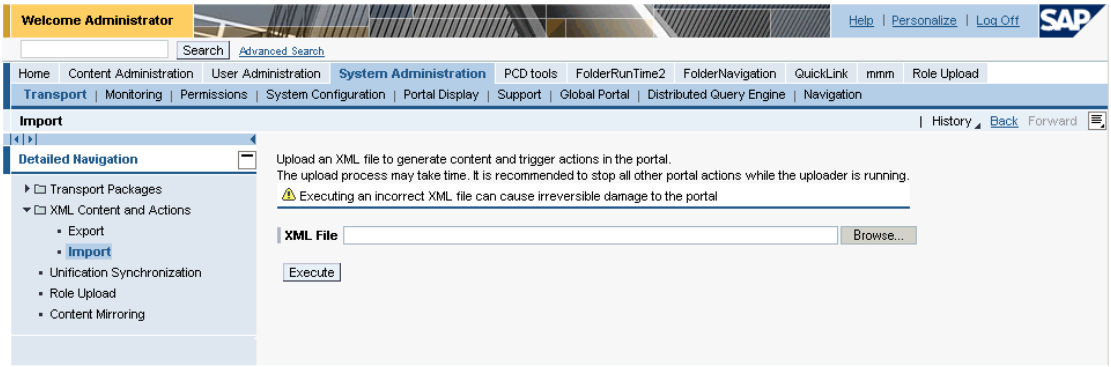
Prerequisites

- A well-formed XML file that is valid for the XML Content and Actions feature.

  Important: It is highly recommended to stop all other portal actions while the import tool is running.

Procedure

1. In the portal, navigate to *System Administration* → *Transport* → *XML Content and Actions* → *Import*.



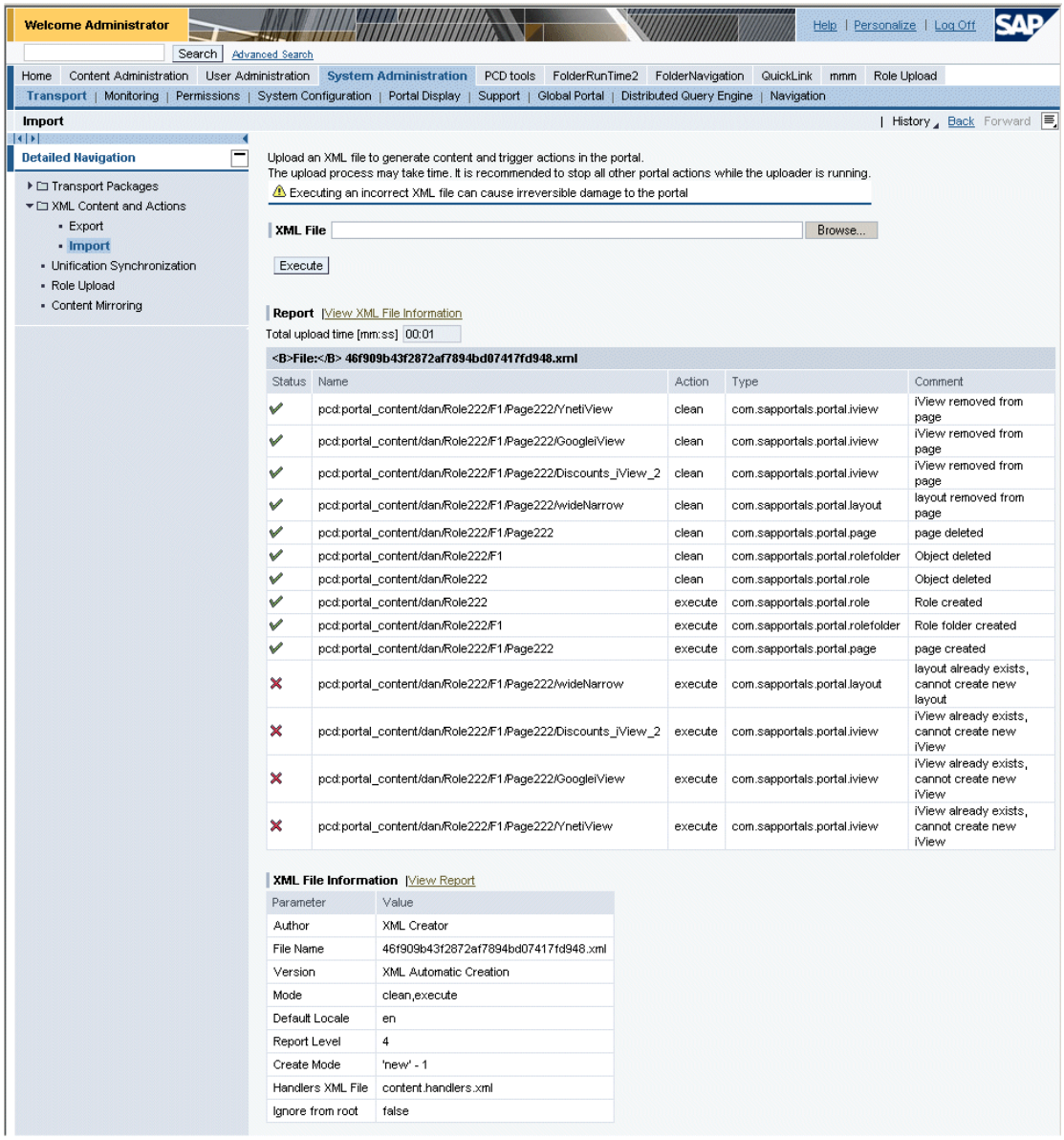
2. Click *Browse*.
3. Locate and select an XML file to import.
4. Click *Execute* to begin the import process. With large files, the process may be time consuming.

   Once you click *Execute*, you cannot stop the import process. All actions are irreversible and there is no rollback feature in case the process aborts in the middle.

   Do not perform any actions in the portal during the import process.

Result

When the script finishes, the results are displayed in a table.



The table contains the following fields:

Field	Description
`Status`	Indicates whether the action was successful. The status can be one of the following: 1. `debug` 2. `info` 3. `warning` 4. `success` 5. `fail` Results are filtered based on each action's status and the report level filter defined in the `report.level` attribute of the `[GenericCreator]` node. Results are displayed for actions with the selected report level and higher. For example, if the report level is `warning`, then actions with a status of `warning`, `success` and `fail` are also displayed. If `debug` is defined, then all result types will be displayed.
`Name`	Specifies the full path and name of the object that was created or modified.
`Action`	Specifies the operation mode as defined in the `mode` attribute of the root element. The value can be one of the following: • **execute:** When a semantic object is created in the PCD, or when an action is performed • **clean:** When a semantic object is deleted in the PCD
`Type`	Specifies the object's class, as specified in the XML.
`Comments`	Provides a summary of the action performed.

General information about the XML file is displayed at the end of the table. If you are at the top of the screen, click *View XML file location* to move directly to this area.