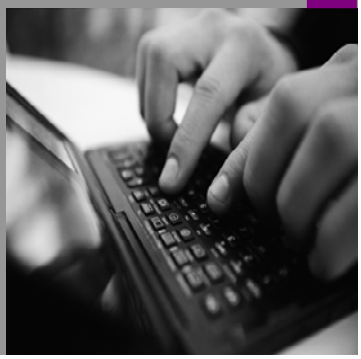


How-to Guide

SAP NetWeaver '04

How To... Upload Content and Actions to the Portal with XML

Version 1.50 – July 2004

Applicable Releases:

SAP NetWeaver '04 (SP Stack 01 and higher)

(SAP Enterprise Portal 6.0 SP3 and higher)

Change Log

Document Version	Comments
1.5	First document release

Contents

1 Upload Content and Actions

Purpose

This document described how you can, through the use of an appropriately coded XML script, automate and simplify the creation of semantic portal objects (for example: iViews, pages, roles, worksets, systems etc.) and perform actions (for example: role assignment and content deletion) in the portal.

A parser in the portal interprets the XML in the uploaded file and generates content and performs actions directly in the Portal Content Directory (PCD) based on this XML.

It can be useful for automated mass content creation, without having to use the wizards and editors available in the portal. Advanced users can use their knowledge of the capabilities and limitations of the tool to expand its uses to other possibilities, such as performing batch operations and pinpoint modifications within a large content base.

Integration

In previous versions of SAP Enterprise Portal 6.0, a service formerly known as the *Generic Creator* was available as an SAP internal tool in the PortalAnywhere environment. The service has been rewritten and is now officially released to customers and available as portal service and iView.

The implementation uses SAP-provided handlers for semantic portal objects and actions; for example, the PageHandler, PackageHandler, and RoleHandler. Customized handlers can be developed by customers to add new functionality or to extend the standard implementation to more than content creation.

Prior to uploading an XML file in the portal, the portal content and action definition needs to be coded into semantically correct XML that can be parsed and executed by the portal. Coding the XML is possible in a number of ways, including the use of scripts that can transform a Microsoft Excel format or a text format to XML. Such services are not supplied by SAP.

Uploading the XML file to the portal is performed via a dedicated iView, which is assigned by default to the standard system administration role. For instructions on using the iView, see “Uploading an XML File to the Portal” on page 28).

Uploaded 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 installation or on test content before using it in live environment.

It is recommended to restrict access to the iView to a limited number of portal administrators who have been trained to use it.

Features

The architecture of this XML-based implementation in the portal is based on the following:

- An engine that is able to parse XML and execute actions, while providing clear and detailed reports to indicate successes and failures of each stage. The engine is modular, thus allowing it to be extended for custom semantic object types.
- A set of handlers, each one dealing with a different semantic object or action.

- A graphical user interface (iView) to allow administrators to upload XML from within the portal using an intuitive and straightforward procedure. This interface is currently referred to as the *Content and Action Upload* tool.

Constraints

- The portal's transport (import/export) mechanism should be used to move content existing in a portal to another portal. The transport mechanism also provides additional functionalities, such as multi-language support.
- The portal does not provide a means to extract existing content in the portal in its current state to XML.
- Currently, there is no editor available in the portal which enables you to view, edit, or validate the syntax and well-formedness of the XML in a file before it is uploaded.

2 Workflow for Upload of Content and Actions

Purpose

This topic summarizes the typical workflow for generating content and actions in the portal by means of an XML file.

Prerequisites

- The semantic objects and actions required in the portal need to be planned and defined before the XML file is coded; for example, in a plain text or Microsoft Excel format.
- Before running the XML script, all portal components on which objects defined by the XML are based, must already be present in the application repository. This includes page layouts.

Process Flow

1. Create the well-formed XML according to the requirements of the XML parser and handlers. Follow the guidelines and syntax described in this document.
2. Upload the XML file via the *Content and Action Upload* tool in the portal.

   The tool is available by default in the portal navigation path: *System Administration* → *Transport* → *Upload Content & Actions*.
3. Review the results of the upload in the user interface of the *Content and Action Upload* tool.
4. Check and test the content in the portal.

3 Architecture and Core Behavior

3.1 Key Components

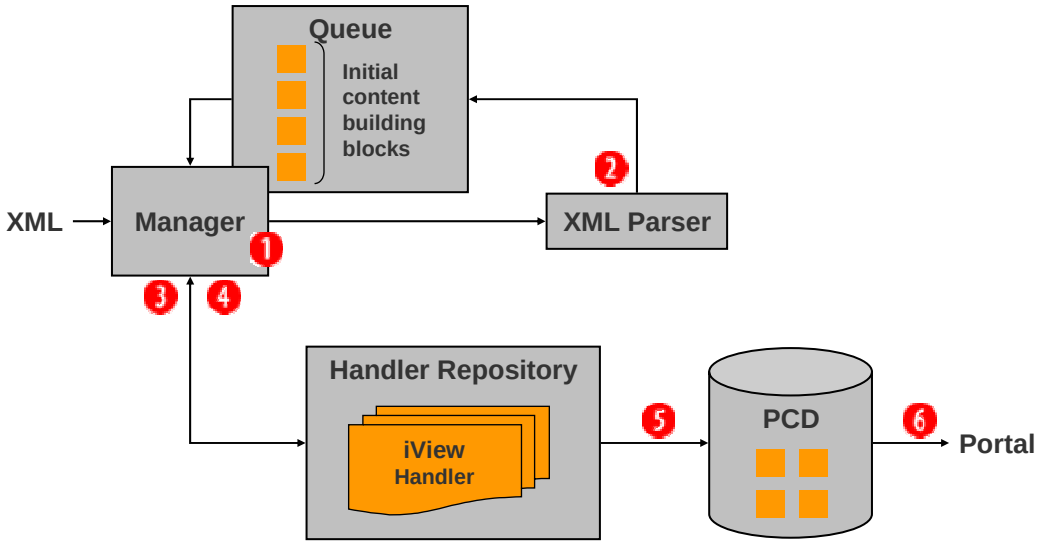
The key components involved in parsing a XML-coded script once it is uploaded to the portal are as follows:

Component	Function
XML Script	• Specifies how the XML should be treated (for example, modes etc.) • Specifies which objects should be created, updated, and deleted • Specifies which actions should be performed
Handlers	• Each semantic objects type and action has its own handler. • Handles the execution of an object or action in the XML script
Generic Creator Engine	• Functions as a process manager • Functions as an XML parser • Associates each building block in the XML script with an appropriate handler • Manages the content creation process with error handling
Content & Action Upload tool (Portal iView)	• Uploads the XML script to the portal for execution by feeding the XML to the Generic Creator Engine.

3.2 Execution Phase

When you upload an XML script in the portal, the following takes place (see also following figure):

1. The XML is checked for semantic correctness and well-formedness. If not, the process is aborted automatically.
2. The parser analyzes the tree structure in the XML hierarchy nodes and determines what should be generated and how, based on global and element-specific parameters. The result is set of building blocks written into an execution queue.
3. The manager loads the required handlers.
4. The manager controls the queue and makes sure each building block is checked by its respective handler. It checks for data sufficiency and determines if the assigned handler can execute the building block. If at least one object or action in the queue fails validation, the entire process is aborted without writing to the PCD.
5. Objects are written to the PCD and actions performed in the portal, based on the XML script.
6. After all the objects and actions have been fully executed the results are displayed on the screen in the form of a detailed report.



Execution phase: This figure illustrates the phases an XML script undergoes from the instant it is uploaded to the portal to the time the respective objects are created and actions performed in the PCD. See the sub-sections before for a description of each step and the various key components.

4 Standard Handlers

Use

The portal relies on handlers developed for each semantic object or action. The following handlers are shipped with the portal:

Handler name*	**Supported semantic objects/actions**
`SystemHandler`	Systems
`FolderHandler`	Folders (in Portal Catalog)
`LayoutHandler`	Page layouts
`PackageHandler`	Packages
`PageHandler`	Portal pages
`PortaliViewHandler`	iViews
`RoleHandler`	Roles
`RoleFolderHandler`	Folders (in role objects)
`WorksetHandler`	Worksets
`DeepCleaner`	Recursive deletion of objects from a designate point *(currently in development)*
`DesktopHandler`	Portal desktops

* The full ID of each handler is:
`com.sap.portal.ivs.genericcreator.handlers.<handler_name>`

For technical information on these handlers, refer to the Javadocs documentation in the Portal Development Kit (PDK).

The above handlers implement the `IHandler` interface either directly or indirectly. For example, during the validation phase before objects are written to the PCD and actions performed in the portal during the execution phase.

5 XML Elements and Attributes

This section describes the elements and attributes required to code a semantically correct XML file that defines semantic objects and actions in the portal and which can be parsed by the Generic Creator Engine.

An XML file must have the following three file elements in order for it to be parsed by the Generic Creator Engine:

- **General details:** Defines general details and configuration settings required by the XML Uploader. These are described in the form of attributes within the start `[GenericCreator]` element tag (see “XML Element: [GenericCreator]” on page 9):
- **Global parameters:** Defines the global properties and values represented by variables throughout the XML file. See “XML Element: [Property]” on page 12.
- **[Context] or [Action] element blocks:** Defines either a semantic object (`[Context]` element) or action (`[Action]` element). See “XML Element: [Context]” on page 13 and “XML Element: [Action]” on page 20.

5.1 XML Element: [GenericCreator]

The `[GenericCreator]` tag element is the root tag of the XML file. Its attributes specify how the XML should be interpreted and determines the behavior of the XML parser.

Definition

The `[GenericCreator]` start-tag is defined in the following manner:

```
<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 table describes the `[GenericCreator]` element attributes:

Attribute	Mandatory	Description and Valid Values
`author`	Yes	Specifies the name of the author of the content in the XML document. Note that the author does not have to be defined as a user in the portal. Although this attribute is required, it has no effect on the portal, nor is related to any background service or operation.

`ignore`	Yes	Specifies whether or not a tag block is executed. This attribute can also be applied as an attribute within the `[Context]` and `[Action]` start tag elements. This attribute is recursive and applies to each building block that is hierarchically below it, unless specified otherwise. For example, if the root `ignore` value in the `[GenericCreator]` tag element is set to **`true`**, then all child building blocks will be skipped, unless `ignore` is specified as **`false`**. Valid values: • `true`: The tag block is not executed. • `false`: The tag block is executed. If this attribute is not specified in the XML document, then the default value is `false`. See also "Executing Specific XML Blocks (Ignore all, except for…)" on page 27.
`report.level`	Yes	Specifies which messages are reported in the Content & Actions Uploader iView in the portal (after an XML file has been uploaded). This attribute does not determine whether or not an XML element is executed. 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 up; 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. Note that an external log file is automatically generated and logs all results, regardless of the report filter level that is defined in the XML file. See "Uploading an XML File to the Portal" on page 28.

`mode`	Yes	Specifies the mode for content creation. Valid values: • `clean`: Objects defined in the XML script within `[Context]` elements are removed from the PCD. All elements of type `[Action]` are ignored. • `execute`: The objects in the XML script are created (without removing them first) in the PCD. Actions declared in the XML are also performed. The value of the `createMode` attribute (see below) determines how the `execute` mode is performed. This attribute can accept a single mode or multiple consecutive modes; if you specify more than one, they must be separated with commas. The order in which they appear is the order in which they will be executed. This means that the entire script will be executed once according to the first mode, and then again according to the following mode, and so on.
`createMode`	Yes	Specifies what action to take when the XML defines an object that already exists in the PCD. Valid only when in *execute* mode (`mode="execute"`). Valid values: • `1`: If the object exists, then **do nothing**. This is the default setting if the attribute is not defined in the XML. • 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 will be updated; (ii) if the XML defines new properties, they will be added to the existing object; and (iii) if the existing object contains properties that are not defined in the XML they will remain unchanged. This attribute is recursive and applies to each building block that is hierarchically below it, unless specified otherwise. For example, if the root `createMode` value in the `[GenericCreator]` tag element is set to **1**, then all child building blocks will be skipped if the objects they define already exist in the PCD, unless `createMode` is specified as **2** as **3**.
`version`	Yes	Specifies the version and/or a short description of the XML document. This attribute is required, but has no effect on the portal, nor is required by any background service or operation.
`default.locale`	Yes	Specifies the default locale for an object if its `[Context]` element does not specify a locale attribute (`originalLocale`). Only required if the `[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">
```

5.2 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 namespaces and default locale settings.

Definition

Each `Property` element attributes defines a single property name-value pair. A `Property` element tag must be nested within the `[GenericCreator]` element. The `Property` element is defined in the following manner.

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

The following table describes the `Property` element attributes:

Attribute	Mandatory	Description and Valid Values
`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 in the `[GenericCreator]` tag element can be used elsewhere in the XML document, in the following manner:

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

Example

```
<!—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"/>
```

5.3 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 determines which type of action is performed on the object: create, delete, or update.

The [Context] element is defined in the following manner.

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

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 are attributes and sub-elements are mandatory. This document describes only the mandatory, basic, and commonly-used attributes and sub-elements. Refer to the Javadocs for each handler for more detailed information. For the list of handlers, see "Standard Handlers" on page 8.

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

Attribute	Mandatory	Description and Valid Values
name	Yes	Specifies the object ID (technical name) of the object. **Note:** Do not specify the full PCD path of the object. An object's PCD location is a concatenation of name values taken from each object that is higher than the current object (within nested [Context] elements). For this reason, nesting multiple [Context] elements is important for generating hierarchy in the PCD.
title	No	Specifies the friendly name of an object. If this attribute is not defined, then the name value 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). It 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. See also "Creating Hierarchies without Nested [Context] Elements" on page 27.

`objectClass`	Yes	Specifies which handler must be loaded in order to process the object. The required handlers are loaded after the entire XML document is parsed. Note that all necessary handlers must be defined in the `context.handlers.xml` file (see “Standard Handlers” on page 8). Valid values (based on standard handlers shipped with the portal): • **`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`** (portal 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.sap.portal.gc.deepCleaner`** (clean action) • **`com.sapportals.portal.desktop`** (portal desktops)

`template`	Yes[1]	Specifies one of the following: • The **source object** to which the current object is related to through a **delta link**. For example, objects based on a template. 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 by any other means. • The **portal component** on which an object is based. This is necessary only if when there are no intermediary objects, such as delta link templates, between the current object and the source code of the iView (portal component). Use the following syntax: **`par:/applications/<PAR_name>/components/<component_name>`** Only iViews, portal pages, page layouts, and system objects can be based on portal component. **Note:** The portal components on which objects are based must be present in the application directory before uploading an XML script. **Note:** This attribute does not specify if the object is an object template in the portal. To define an object as a template, use the `IsTemplate` property in the object (you can also do this in the with the `[Attribute]` and `[AttributeValue]` elements.

`create_as`	Yes	Specifies the relationship of the object being created to a template or portal component on which it is based. This attribute is dependent on the `template` attribute (see previously).
		Valid values (for code samples, see “Creating an iView” on page 22):
		• **`0`**: Assign this value in the follow instances:
		  ○ To create a new object that is based directly on portal component. The `template` attribute specifies which portal component.
		  ○ To make a copy of an object that already exists in the PCD. The current object being created and the original one being copied become siblings and share the same source object (via a delta link) or portal component. The `template` attribute specifies the source object.
		• **`1`**: Creates an object that is a delta link to the object specified in the `template` attribute).
		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 respective values from their source objects. To assign different object properties, use the `[Attribute]` and `[AttributeValue]` elements (see later in this document).

`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, page can only have a single default page layout. If you assign more than one default layout to a page, then the last one in the script will be designated the default layout. 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. Note that this attribute is necessary for SAP internal content developers only in order to support in-house translation. Do not use the value: **EP**, as it is reserved for SAP content.
`collection`	No	Specifies the collection setting of the object. Note that this attributes is necessary for SAP internal content developers only in order to support in-house translation. Do not use the value: **IP_PTL_INITIAL_CONTENT**, as it is reserved for SAP content.
`ignore`	No	This attribute functions in an identical manner to the same attribute defined in the root `[GenericCreator]` element. For more information, see “XML Element: [GenericCreator]” on page 9. If you define a value for this attribute in the `[Context]` element start-tag it will override the value defined in the `[GenericCreator]` element. However, if no value is defined in the `[Context]` element, the value defined in the `[GenericCreator]` element will be used instead.
`createMode`	No	This attribute functions in an identical manner to the same attribute defined in the root `[GenericCreator]` element. For more information, see “XML Element: [GenericCreator]” on page 9. If you define a value for this attribute in the `[Context]` element start-tag it will override the value defined in the `[GenericCreator]` element. However, if no value is defined in the `[Context]` element, the value defined in the `[GenericCreator]` element will be used instead.

[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.

The following list specifies common XML elements that are commonly nested with the `[Context]` element.

- `[Attribute]`
- `[AttributeValue]`

See “XML Element: [Attribute] and [AttributeValue]” on page 18.

Usage

To assign one object to another, for example an iView to a page, you must nest the `[Context]` element of the child object within the `[Context]` element of the parent object. Note that the `parent` attribute defined in a root `[Context]` element can be used as an alternative method to nest an object within another object in the PCD.

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 respective editor.

Example

See “XML Code Samples for Content Creation” on page 22.

5.4 XML Element: [Attribute] and [AttributeValue]

The [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, object properties and values are already defined with portal components (in PAR files). Thus, you would generally use the [Attribute] and [AttributeValue] elements in the following instances:

- 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.
- Assign values to existing properties that are not initially assigned a value.
- Define new properties if necessary (a rare scenario)

The [Attribute] and [AttributeValue] elements can also be used to pass on information to a handler in order to perform a particular action or determine a policy of some type.

XML Element	Description
[Attribute]	Specifies the metadata of the property, such as name and value type.
[AttributeValue]	Specifies the actual value of the property.

Definition

An [Attribute] element tag must be nested within the [Context] element to which it relates. The [AttributeValue] element must be nested within the [Attribute] element to which it relates.

An [Attribute] and [AttributeValue] element tag structure is defined in the following:

```
<Attribute name="<attribute_name>" type="<attribute_type>"
isOrdered="<attribute_ordering>"
Inheritance="<attribute_inheritance>" WritePermission="IGNORE_ACL">

      <AttributeValue value="<attribute_value>" locale="<locale>"
      />

</Attribute>
```

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

Attribute	Mandatory	Description and Valid Values
`name`	Yes	Specifies the name of the property variable.
`type`	Yes	Specifies the expected data type of the property's value. Valid values: • **`string`** • **`text`** • **`integer`** • **`boolean`** • **`double`**
`Inheritance`	Yes	Specifies the inheritance mode of the property. This property is currently not supported by the portal. In the meantime, you define the following value: **`NONFINAL`**

The following table 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`.

Usage

The Portal Development Kit (PDK) should be able to assist you to determine the names of basic properties for most semantic object types.

Example

```
<Context>
   ...
   <Attribute name="com.sap.portal.pcm.Description" type="text"
isOrdered="true" Inheritance="NONFINAL"
WritePermission="IGNORE_ACL">
      <AttributeValue value="Schedule Processing" locale="en" />
   </Attribute>
</Context>
```

5.5 XML Element: [Action]

XML elements of type [Action] differ in concept and syntax to XML elements of type [Context]. As the name of the XML element implies, [Action] performs an action within the portal, instead of generating or updating a semantic object in the PCD. Actions are typically general; they tend to be not specific to a particular object type (although it is possible to develop a handler of type [Action] that operates on a particular content type).

The following table describes the [Attribute] element attributes:

Attribute	Mandatory	Description and Valid Values
id	Yes	Specifies which handler must be loaded in order to process the specified action. Note that all necessary handlers, including standard and custom-made, must be defined in the context.handlers.xml file (see “Standard Handlers” on page 8).
ignore	No	This attribute functions in an identical manner to the same attribute defined in the root [GenericCreator] element. For more information, see “Standard Handlers” on page 8. If you define a value for this attribute in the [Action] element start-tag it will override the value defined in the [GenericCreator] element. However, if no value is defined in the [Context] element, the value defined in the [GenericCreator] element will be used instead.

Definition

A typical [Action] element tag structure is defined in the following manner:

```
<Action id="<handler_name>" ignore="<mode>" <additional attributes specific to each handler> />
```

Usage

The sub-sections that follow describe how to use the supported actions.

Note the following:

- [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 (through the mode attribute specified in the [GenericCreator] root element). Note that the createMode attribute is disregarded by [Action] elements.

5.5.1 Action ID: [DeepCleaner]

`[DeepCleaner]` 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 the *DeepCleaner* handler.

Attribute	Mandatory	Description and Valid Values
`id`	Yes	Enter the following value: **`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.

Definition

A typical `[Action]` element tag using the *DeepCleaner* handler is defined in the following manner:

```
<Action id="com.sap.portal.gc.deepCleaner" ignore="<mode>"
root.folder="<folder_id>" exclude.folder="<absolute_folder_id>" />
```

Example

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

5.6 XML Code Samples for Content Creation

This section includes some basic XML code samples for creating basic portal objects using the *Content & Action Uploader* tool.

5.6.1 Creating a Folder in the Portal Catalog

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

5.6.2 Creating an iView

iViews, portal pages, and systems can be based directly on portal components or on other objects though delta links. These relationships can be created in a number of ways using the appropriate XML scripting and usage of attributes (particularly `template` and `create_as`).

The following examples show you can create an iView based on either a portal component or a template.

5.6.2.1 Based on Portal Component

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

See iView 1 in the following figure.

5.6.2.2 Based on iView (Example 1: Delta Link)

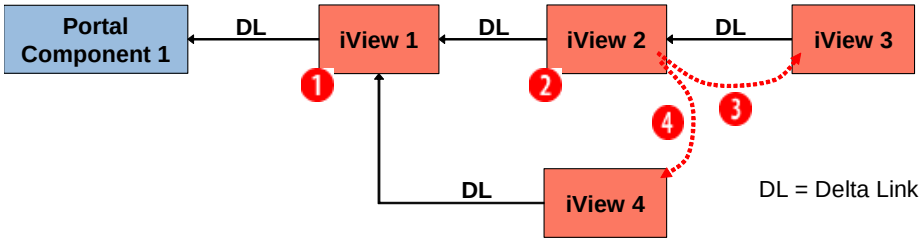
```
<Context name="${namespace}.contentCatalog"
template="portal_content/com.sap.pct/admin.templates/iviews/${na
mespace}.contentCatalog"
objectClass="com.sapportals.portal.iview" create_as="1"
container="com.sap.portal.reserved.layout.Cont2">
 <Attributes>
    <Attribute
name="com.sap.portal.reserved.iview.IsolationMode"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

See iView 2 in the following figure.

5.6.2.3 Based on iView (Example 2: Copy)

```
<Context name="${namespace}.contentCatalog"
template="portal_content/com.sap.pct/admin.templates/iviews/${na
mespace}.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"
isOrdered="true">
       <AttributeValue value="URL"/>
    </Attribute>
    <Attribute name="com.sap.portal.iview.HeightType"
type="string" isOrdered="true">
       <AttributeValue value="FULL_PAGE"/>
    </Attribute>
    <Attribute name="com.sap.portal.iview.ShowTray"
type="string" isOrdered="true">
       <AttributeValue value="false"/>
    </Attribute>
 </Attributes>
</Context>
```

See iView 3 in the following figure.



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"

This figure illustrates 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 defined for each iView in the XML.

5.6.3 Creating a Portal Page

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

5.6.4 Creating a Portal Layout

```
<Context name="${namespace}.fullWidth"
template="par:/applications/com.sap.portal.layouts.default/compo
nents/fullWidth" objectClass="com.sapportals.portal.layout"
create_as="0" collection="${collection}" domain="EP"
originalLocale="${locale}"/>
<Context name="${namespace}.portal_information"
template="portal_content/com.sap.pct/admin.templates/pages/${nam
espace}.portalpagetemplate"
objectClass="com.sapportals.portal.page" create_as="1"
title="Portal Information">

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

</Context>
```

5.6.5 Creating a Workset

```
<Context name="${namespace}.home.company"
objectClass="com.sapportals.portal.workset" entryPoint="false"
asUnit="true" title="Company" collection="${collection}"
domain="EP" originalLocale="${locale}">
 <Attributes>
    <Attribute name="com.sap.portal.navigation.MergeId"
type="string" isOrdered="true">
       <AttributeValue value="${namespace}.home.company"/>
    </Attribute>
 </Attributes>
</Context>
```

5.6.6 Creating a Role

```
<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"
isOrdered="true">
       <AttributeValue value="true"/>
    </Attribute>
 </Attributes>
</Context>
```

5.6.7 Creating a Role Folder

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

5.6.8 Creating a System

Note that it is not possible to define in the XML script an “alias” for a system object.

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

5.6.9 Assigning an iView to a Portal Page

```
<Context name="${namespace}.batchUpload"
template="portal_content/com.sap.pct/admin.templates/pages/${nam
espace}.portalpagetemplate"
objectClass="com.sapportals.portal.page" create_as="1"
title="User Data Import">
    <Context name="${namespace}.fullWidth"
template="portal_content/com.sap.pct/admin.templates/layouts/${n
amespace}.fullWidth" objectClass="com.sapportals.portal.layout"
create_as="1" PrimaryLayout="true">
    </Context>

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

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

5.7 Tips and Tricks

5.7.1 General Information

- Avoid using special characters in the object ID of content objects.
- Use the correct data types and locale for properties in content objects.

5.7.2 Executing Specific XML Blocks (Ignore all, except for…)

If you use an XML script to create mass content in your portal, and want to re-use it to make any number of pinpoint changes to content that has already been created using the file, you can apply the `ignore` attribute in appropriate places to skip all blocks in the XML document, except for those you want to execute. This is possible due to the recursive behavior of the ignore attribute, which applies itself to each building block that is hierarchically below it, unless specified otherwise.

For example, configure the XML document as follows:

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

   Note:

   To apply additional pinpoint changes to existing objects, you can use the `ignore` attribute in conjunction with the `createElement` attribute in the reloaded XML.

This procedure supports both `mode` types: `execute` and `clean`.

5.7.3 Creating Hierarchies without Nested [Context] Elements

Typically, you nest one `[Context]` element within another to generate object hierarchies in the PCD. However, by using the `Parent` attribute in the `[Context]` element, you can nest and object within another on the level of the PCD without nesting XML elements. 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. In other words, you cannot use it in a `[Context]` element that is nested in another `[Context]` element.

The following example shows the use of a `Parent` attribute to create a nested iView.:

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

6 Uploading an XML File to the Portal

Use

You use the Content and Action Uploader iView to upload to the portal a well-formed XML file, which describes portal semantic objects and actions that are based on the rules and guidelines described in this guide.

Prerequisites

- A semantically correct XML file whose syntax adheres to the requirements of the portal's XML parser implementation.
- It is highly recommended to stop and prevent all other portal actions while the uploader is running.

Procedure

1. In the portal, navigate to *System Administration* → *Transport* → *Upload Content & Actions*.
2. Click *Browse*.
3. Locate and choose the XML file containing the content and action description.
4. Click *Execute* to begin the upload process. Depending on the size of the XML file the upload process may be time consuming.

   Once you execute the upload it cannot be aborted until it is finished. All actions are irreversible and there is no rollback in the event the process is aborted midway.

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

Result

For a detailed description of the execution phase, see "Architecture and Core Behavior" on page 6. Upon completion of the upload, the results are displayed on the screen in a tabular form.

The results table displays the following:

Output	Description
Status	Specifies the status of the action performed. Note the results are filtered according to the report level filter defined in the XML file (`report.level` attribute). The default report levels implemented by the standard handlers are as follows: 1. debug 2. info 3. warning 4. success 5. fail Results are displayed from the selected report level and up; for example, if `success` is defined as the report level, then results of type `success`, `warning` and `fail` will also be displayed. If `debug` is defined, then all result types will be displayed.
Name	Specifies the full path of the object and its name.

Action	Specifies the operation mode (`mode` attribute) as defined in the XML file: • **execute:** used when a semantic object is created in the PCD, or when an action is performed (for example, role assignment) • **clean:** used when a semantic object is deleted in the PCD
Type	Specifies the object class used to execute the process (as specified in the XML).
Comments	Provides a summary of the action performed.

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