Accessing the PCD

HELP.NW_DEVGUIDE_EP_PCD

Release 663

Copyright

Icons in Body Text

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help* → *General Information Classes and Information Classes for Business Information Warehouse* on the first page of any version of *SAP Library*.

Typographic Conventions

Type Style	Description
Example text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation.
Example text	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
Example text	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
<Example text>	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

Accessing the PCD

The Portal Content Directory (PCD) is the main repository for portal content, both delivered with the portal and created by administrators. The PCD contains a hierarchy of folders, each of which can contain semantic objects, such as iViews, pages and roles.

The PCD is stored in the portal database and is accessed via the PCD Generic Layer (GL), a JNDI provider that implements additional functionality, including the following:

- **Personalization:** The PCD enables portal objects to be personalized. For each attribute of each portal object, different values can be stored for each user.

  For example, an iView that displays the weather can have an attribute called `City`. Each user can set the `City` attribute for that iView to a different city, so the weather for that city is displayed for that user.

- **Delta Links:** The PCD enables the creation of portal objects whose attributes are inherited from another portal object. Changes on the original object update the delta links.

  For more information, see Delta Links [External].

- **Locking:** The PCD enables the locking of objects to avoid concurrent modifications.

This section provides the following:

- Overview [Page 7]: Describes what occurs when a PCD lookup is performed.
- Working with Semantic Objects [Page 15]: Describes how to work with interfaces for specific semantic objects, such as iViews and pages.
- Working with Administration (PCM) Objects [Page 35]: Describes how to work with common interfaces implemented by all portal objects.
- Working with PCD Objects [Page 42]: Describes how to work with low-level interfaces.

Dependencies

The following are the dependencies for using the classes and interfaces described in this section:

Semantic Objects

Sharing References:

- `com.sap.portal.ivs.api_iview`
- `com.sap.portal.ivs.api_landscape` (for `ISystem`, `ISystems`)

JAR files:

- `com.sap.portal.ivs.api_iview_api.jar`
- `com.sap.portal.ivs.api_landscape_api.jar` (for `ISystem`, `ISystems`)

Administration (PCM) Objects

Sharing References:

- `com.sap.portal.pcm.admin.apiservice`
- `com.sap.portal.pcd.basicrolefactory` (for role assigner permission constant)

JAR files:

- `com.sap.portal.pcm.admin.apiservice_api.jar`
- `com.sap.portal.pcd.basicrolefactory_api.jar` (for role assigner permission constant)

PCD

Sharing References:

- `com.sap.portal.pcd.glservice`

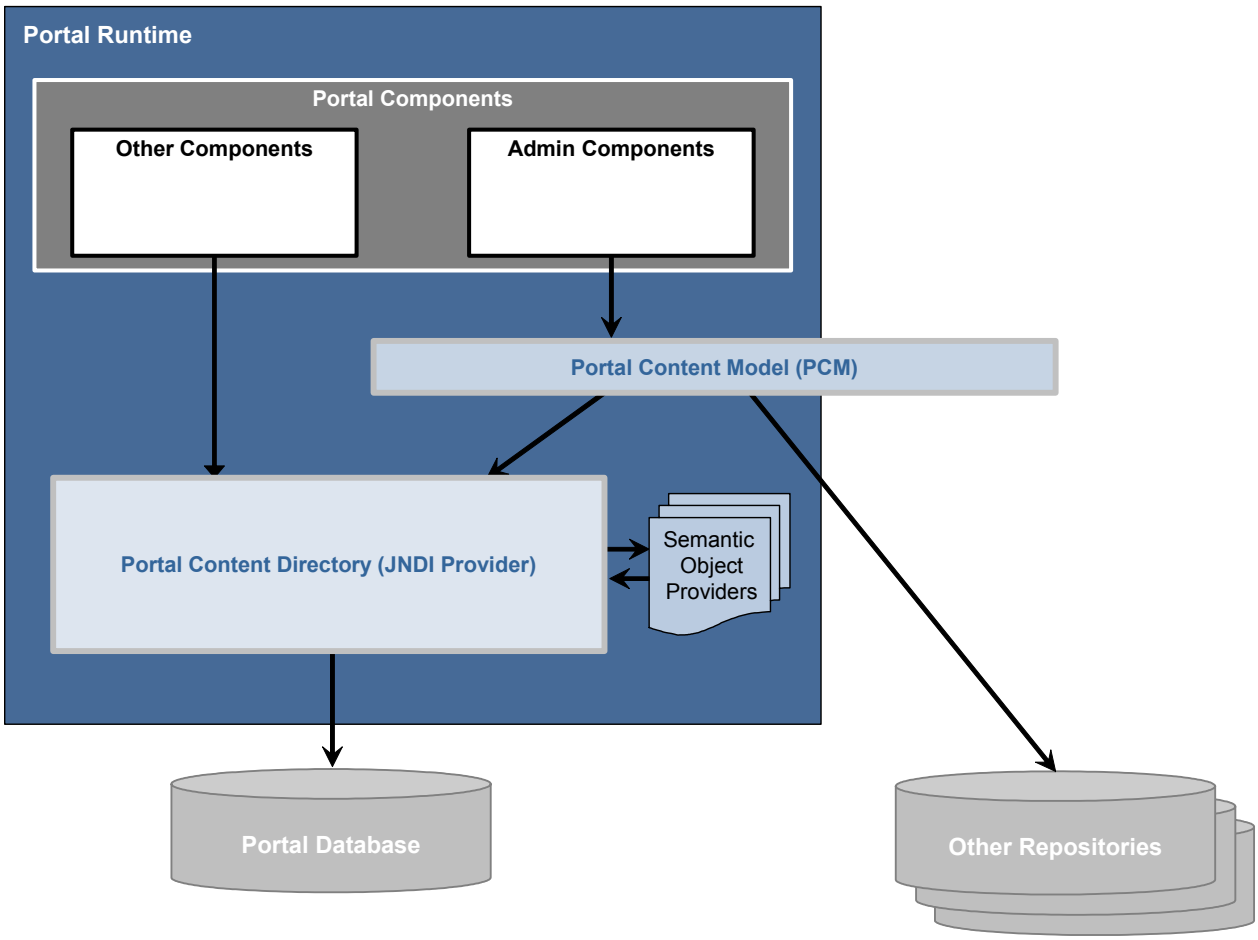
JAR files:

- `com.sap.portal.pcd.glservice_api.jar`
- `gl_api.jar`

Overview

The PCD stores portal objects, and provides an API to enable applications to perform lookups and modify the PCD.

The following shows the major components involved when working with the PCD:



- **Portal Database:** Stores portal objects.
- **Portal Content Directory (PCD):** A JNDI provider that provides an API for querying portal objects.
- **Semantic Object Providers:** Each provider defines the logic for returning Java objects when querying a specific portal semantic object. A semantic object is defined by its `com.sap.portal.pcd.gl.ObjectClass` property.
- **Portal Content Model (PCM):** A set of common interfaces for all portal objects.

  Administration editors, for example, request PCM interfaces when querying the PCD and other repositories for portal objects. These editors need higher-level interfaces than those provided by the PCD, but do not need the specific interfaces for the different semantic objects.

  For more information, see Portal Content Model (PCM) [Page 14].

An application can look up a PCD object and request that a variety of Java interfaces be returned. For more information on what occurs when an application performs a PCD lookup, see How PCD Lookups are Performed [Page 8].

How PCD Lookups are Performed

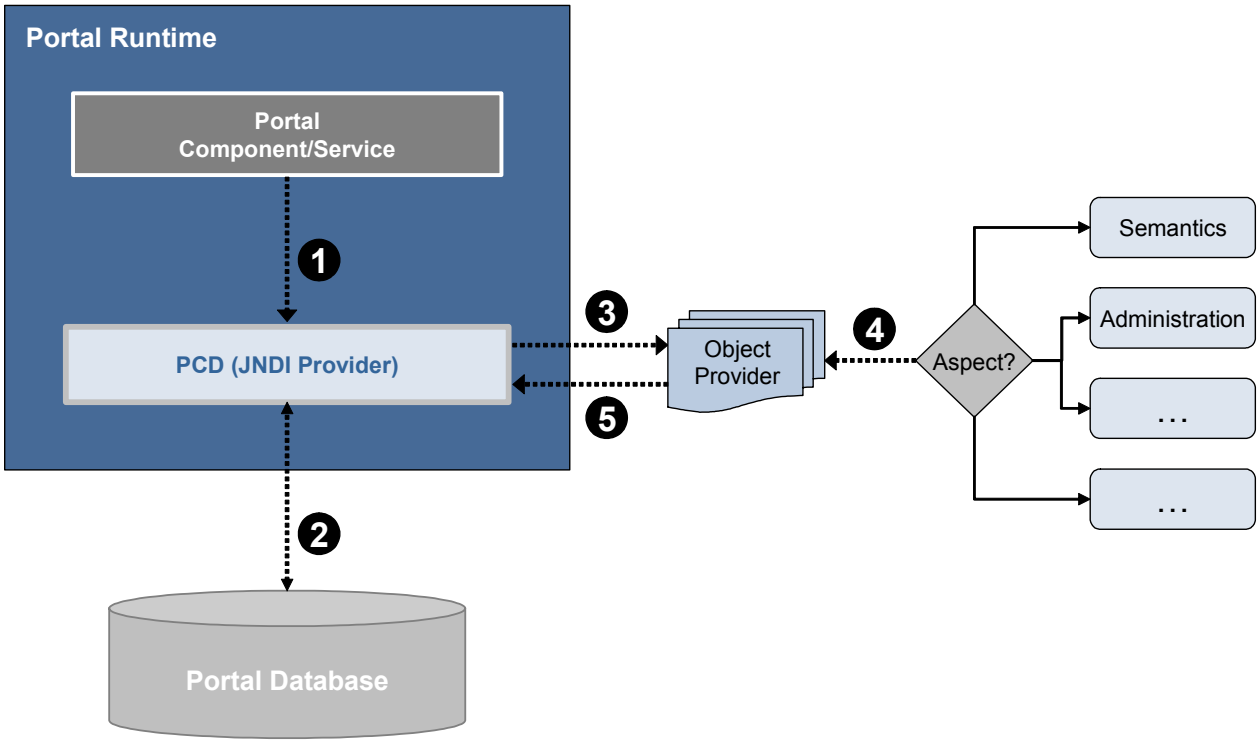
In the PCD, all objects are simply contexts with a set of attributes.

One attribute is the object class (`com.sap.portal.pcd.gl.ObjectClass`), which indicates the type of object. When a lookup is performed on an object, the PCD checks the object class attribute to determine which object provider to use for creating a Java object from the set of attributes.

The object class `com.sap.portal.pcd.gl.GlContext` indicates a folder object, which does not have a special object provider. All other values specify a semantic object.

Process Flow

The following describes how PCD lookups are performed:



1. A portal component or service creates a PCD initial context, including specifying the following environment variables:
   a. **Initial Context Factory**
   b. **Aspect:** Indicates to the object factory what type of Java object to return.

      For more information, see Aspects [Page 11].
   c. **User:** PCD checks if this user has permissions to perform this lookup.
   d. **Personalization:** Indicates whether to get a personalized object and, if so, for which user. This variable is optional.

      For more information, see Personalization [Page 52].

The following is an example of setting the environment variables (`env` is a `Hashtable` that holds the environment variables, and `request` is the `IPortalComponentRequest` for the current request):

```
import javax.naming.Context;
import com.sapportals.portal.pcd.gl.IPcdContext;
import com.sap.portal.directory.Constants;
import com.sapportals.portal.pcd.gl.IPcdAttribute;

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Constants.REQUESTED_ASPECT,
IPcdAttribute.PERSISTENCY_ASPECT);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(IPcdContext.PCD_PERSONALIZATION_PRINCIPAL,
request.getUser());
```

With the PCD initial context, the application requests a lookup, by calling `lookup()` on the initial context and providing a PCD name. For more information, see PCD Name [Page 12].

2. The PCD checks that the name is valid.
3. The PCD finds the object type of the object being looked up, and calls the appropriate object provider for that type of object. For more information, see Semantic Objects [Page 10].

   If the aspect was set to `IPcdAttribute.PERSISTENCY_ASPECT`, the PCD does not call the object provider but, instead, creates and returns a `IPcdContext` object, and the process stops here.
4. The object provider checks the aspect, and creates the appropriate object for that aspect.
5. The object is returned to the PCD provider, which returns it to the calling application.

Semantic Objects

`IPcdContext` is the low-level representation of all objects in the PCD, similar to the standard JNDI `Context` interface. However, the PCD enables the creation and registration of semantic objects, along with object and state factories, so that a PCD lookup can return different Java objects for different types of PCD objects.

A PCD object's type is defined by the attribute `com.sap.portal.pcd.gl.ObjectClass`. For example, an iView has a value of `com.sapportals.portal.iview` for this attribute.

Object Factories

When performing a lookup, one of the following occurs:

- The PCD automatically determines the semantic object factory for the current object. The semantic object factory determines what type of Java object to return, generally based on the aspect set for the current lookup.

  For more information, see Aspects [Page 11].

- You specify that you want the default PCD object factory, in which case an `IPcdContext` object is returned.

Aspects

The aspect of a PCD lookup is an environment variable that specifies the type of object that should be returned when performing a PCD lookup.

For example, when looking up a page, you may want to return an `IPage` object, which contains methods related to a page, such as for adding iViews to the page or setting the layout. Instead, you may want an `IAttributeSet`, which provides a generic interface that provides access to all the administrative attributes of the page, whether they are defined in the PCD or elsewhere.

The aspect is set with the `com.sap.portal.directory.Constants.REQUESTED_ASPECT` environment variable. It is up to the object provider for the type of object that is requested to read this value and return an appropriate object.

The following table shows the available aspects, and the interfaces that are returned for each:

Aspect	Value of Environment Variable	Object Types Returned

Semantic	`PcmConstants.ASPECT_SEMANTICS`	`IiView`
		`IPage`
		`ILayout`
		`ISystem`

Administration	`PcmConstants.ASPECT_ADMINISTRATION`	`IAdminBase`
		From the `IAdminBase` interface, you can also derive the following interfaces with the `getImplementation()` method:
		`ICatalogNode`
		`IPermission`
		`IAttributeSet`

Persistence	`IPcdAttribute.PERSISTENCY_ASPECT`	`IPcdContext`

If no aspect is specified or an aspect is not recognized by an object factory, the object factory returns the default object. For folders, this is an `IPcdContext`; for semantic objects, this is the Java semantic object.

PCD Name

Each object in the PCD is referred to by its PCD name, which is the full path to the object through the PCD tree structure. For example, the following PCD name,

```
portal_content/myFolder/stocks
```

points to a PCD object called *stocks*, which is in a folder called *myFolder*, which is in the top-level folder *portal_content*.

Atomic Name

The atomic name of a PCD object is the name of the object, without the entire path.

In the example above, the name *stocks* is the object's atomic name, which must be unique within the folder that contains the object (in this case, *myFolder*).

PCD Prefix

The PCD name sometimes is written with the prefix `pcd:`, so that the above PCD name would be written:

```
pcd:portal_content/myFolder/stocks
```

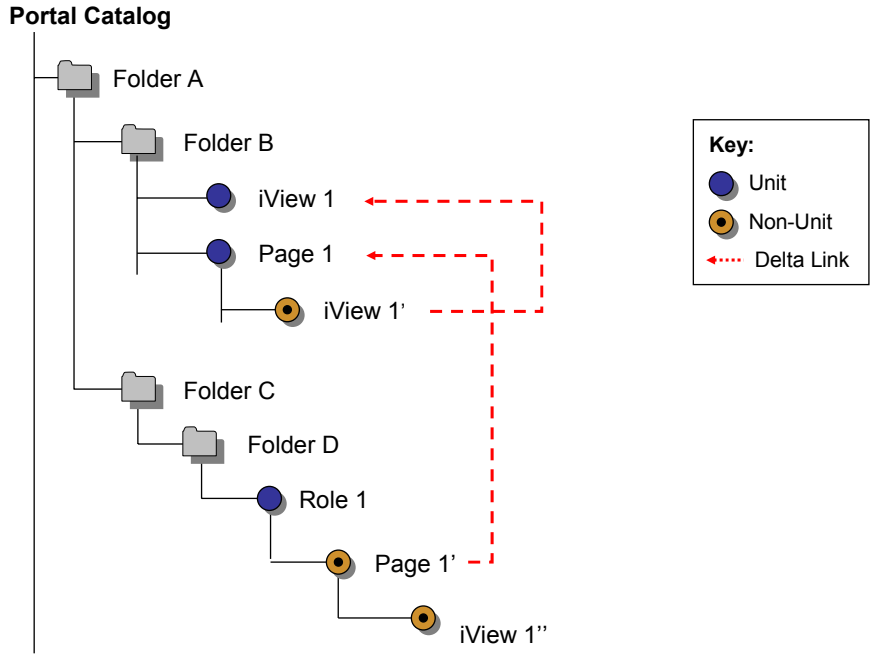
When performing a PCD lookup, you can specify the PCD initial context in one of the following ways:

- Set the `Context.INITIAL_CONTEXT_FACTORY` environment variable to `IPcdContext.PCD_INITIAL_CONTEXT_FACTORY` when creating an initial context.
- When calling `lookup()` on an initial context, add the `pcd:` prefix to the PCD name of the object you are looking up.

Units

PCD content is organized into units, which are semantic objects (such as iView, page and role) whose parent hierarchy consists only of plain folders. A unit includes the semantic object's complete child hierarchy, and the semantic object is called the root of the unit.

For example, a page in a folder in the top-level Portal Content folder is a unit, but an iView in the page or a delta link of the page in a role are not units.



Many tasks can only be performed on units and not on objects within a unit. The following features apply only to unit objects:

- **Permissions:** Only unit objects and plain folders have ACLs. Objects inside a unit object inherit their ACLs from the unit object.
- **Delta Links:** Delta links can only be created from unit objects.
- **Transport:** Only unit objects can be transported.
- **Translation:** Translation worklists include only unit objects.
- **Caching:** Only unit objects can be cached.

Portal Content Model (PCM)

The Portal Content Model is a set of interfaces that enables the creation of an abstraction layer above the PCD. This serves the following purposes:

- For administrative purposes, such as creating and maintaining objects or assigning permissions, it is advantageous to have a set of common semantic interfaces that are implemented by all semantic objects.
- Since some portal objects are housed in locations outside the PCD, the portal needs an abstraction layer for calling portal repositories and returning portal-relevant objects, without depending on the PCD.
- Instead of requiring developers to work with relatively low-level interfaces, such as `javax.naming.directory.DirContext` or the PCD-specific `com.sapportals.portal.pcd.gl.IPcdContext`, the PCM provides more relevant and standard interfaces that are applicable to all types of portal objects.

JNDI calls to the PCD that request the administration aspect are required to provide PCM, or portal administration, objects. For information on aspects and the objects that are returned for each, see Aspects [Page 11].

For information on administration objects, see Working with Administration Objects [Page 35].

Working with Semantic Objects

A semantic object is a persistable portal object, such as an iView or page. Semantic objects are persisted in the PCD as a collection of attributes. By creating and modifying these objects in the PCD, you can change the iViews, pages and other portal objects that are displayed in the portal or that are available to administrators.

In order to obtain semantic Java objects from a PCD lookup, set the environment variable `Constants.REQUESTED_ASPECT` to `PcmConstants.ASPECT_SEMANTICS`. For more information, see Aspects [Page 11].

Java Interfaces

For each type of semantic object, there are generally two types of Java interfaces:

- The semantic object interface, such as `IiView` for an iView.

  Create an instance of a semantic object interface by performing a JNDI lookup, as described in Looking Up Objects [Page 19].

- A helper service interface that provides helper methods for working with that type of semantic object, such as `IiViews` for iViews.

  Some helper services simply provide special implementations of the methods defined in the `com.sap.portal.pcm.IObjectsManager` interface, which these helper services implement. Others provide additional methods. For example, `IiViews` defines no new methods, while `ISystems` provides additional methods, for example, for retrieving all systems defined in the PCD.

  Create an instance of a helper service as you would any portal service, with the `PortalRuntime` class. The following shows how to create the helper service for iViews:

```
IiViews iViewSrv = (IiViews)PortalRuntime.getRuntimeResources()
   .getService(IiViews.KEY);
```

The following are the public Java semantic interfaces:

Semantic Object	Interfaces
iView	`IiView, IiViews`
Page	`IPage, IPages`
Layout	`ILayout, ILayouts`
System	`ISystem, ISystems`

All Semantic Objects

This section describes how to perform the following tasks for all semantic objects:

- Creating Objects [Page 17]
- Looking Up Objects [Page 19]
- Getting/Setting Attributes [Page 20]
- Deleting Objects [Page 21]

Some updates to a semantic object require that you call `save()` on the affected semantic object (such as when updating attributes), while other changes do not require a call to `save()` (such as when adding iViews to a page). Check the Javadocs to determine if a call to `save()` is required.

Creating Objects

This section describes how to create a semantic object in the PCD by first creating a descriptor for the new object and binding it to a folder context.

A new object can be based on an existing PCD object (either as a copy or a delta link) or on an application deployed to the J2EE server.

Procedure

1. Retrieve an instance of the helper object for the type of semantic object that you want to create, such as, `IiViews` for an iView, `ISystems` for a system, and so forth.

```
IiViews iViewSrv = (IiViews)
    PortalRuntime.getRuntimeResources().getService(IiViews.KEY);
```

2. Create a descriptor for the new object that you want to create.

```
INewObjectDescriptor IVtoCreate = (INewObjectDescriptor)
    iViewSrv.instantiateDescriptor(CreateMethod.NEW,
        "par:/applications/myProject/components/myComponent",
            request.getUser());
```

`instantiateDescriptor()` takes the following parameters:

- **New or Delta Link:** Indicates whether the new object is a copy of or a delta link to an existing object. Use constants from the `CreateMethod` class.
- **PCD Object or Application:** Indicates the existing PCD object or portal application on which to base the new object. The format can be one of the following:
  - **PCD name**, such as `pcd:portal_content/myFolder/myObject`
  - **Portal Component name**, in the following format:

    `par:/applications/myApp/components/myComp`

    where *myApp* is the name of an application and *myComp* is the name of a component in *myApp*.

  If you specify a PCD name, the first parameter can be either `NEW` or `DELTA_LINK`. If you specify an application, the first parameter must be `NEW`.
- **Current User:** An `IUser` object for the user making the JNDI call.

3. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

4. Perform a lookup of the folder in which you want to create the new object.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);

    String folderID = "pcd:portal_content/myFolder";
    Context ctx = (Context)iCtx.lookup(folderID);

...
```

5. Create the object by binding the descriptor for the new object to the folder context.

```
...

    ctx.bind("myNewHelloIV", IVtoCreate);

}
catch(NamingException e)
{
}
```

Looking Up Objects

This section describes how to look up a PCD object and get the semantic object for the object.

Procedure

1. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

2. Perform the lookup by supplying the PCD name of the object, and then cast the returned object to the appropriate semantic object interface.

```
InitialContext iCtx = null;
try
{
    String iViewID = "pcd:portal_content/myFolder/stocks";

    iCtx = new InitialContext(env);
    IiView myIView =(IiView)iCtx.lookup(iViewID);
}
catch(NamingException e)
{
}
```

Getting/Setting Attributes

This section describes how to get and set object attributes using the Java semantic interfaces.

For more information on getting/setting attributes using administration interfaces, see Attributes [Page 37]. For more information on getting/setting attributes using PCD interfaces, see Contexts and Attributes [Page 43].

Procedure

In the following examples, `obj` is a semantic Java object. For more information on retrieving an object, see Looking Up Objects [Page 19].

- **Getting Attributes and Meta-Attributes**

```
response.write(obj.getAttribute("attribute"));

response.write(obj.getMetaAttribute("attribute", "meta-attribute"));
```

Text attributes generally require an additional parameter that indicates the locale.

The following attributes require the use of the locale:

- `com.sap.portal.pcm.Title`
- `com.sap.portal.pcm.Description`

The following meta-attributes require the use of the locale:

- `plainDescription`
- `longDescription`
- `category`
- `validValueTitle0`, `validValueTitle1`, and so forth.

```
response.write (obj.getAttribute ("attribute", request.getLocale() )
);
```

- **Setting Attributes**

```
obj.putAttribute("attribute","value");
obj.save();
```

Attribute Constants

To specify an attribute, use the designated constant for that attribute. The constants for each semantic type are located in a corresponding interface in the `com.sap.portal.pcm.attributes` package.

For example, the constants for iView attributes are located in the `IAttriView` interface. The following code checks whether the current iView allows browser caching:

```
response.write(
    obj.getAttribute(IAttriView.ATTRIBUTE_ALLOW_BROWSER));
```

Deleting Objects

This section describes how to delete a semantic object from the PCD.

Procedure

1. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

2. Perform a lookup of the folder that contains the object that you want to delete.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);

    String folderName = "pcd:portal_content/myFolder";
    Context ctx = (Context)iCtx.lookup(folderName);

...
```

3. Delete the object by unbinding the object from the folder that contains it. Use the atomic name of the object.

```
...

    String atomicName = "myObject";
    ctx.unbind(atomicName);

}
catch(NamingException e)
{
}
```

You can instead perform an unbind directly on the initial context, and supply the full path to the object.

iViews

This section describes the tasks that can be performed for iView semantic objects:

- Adding Related Items [Page 23]

For information on creating, modifying and deleting iViews and other semantic objects, see All Semantic Objects [Page 16].

Adding Related Items

This section describes how to associate iViews with other iViews or pages, so that when an iView is displayed, links to its related iViews and pages are displayed in the detailed navigation panel.

The following types of related items can be created:

- **Related Links:** Links to the related iViews and pages are displayed in the Related Links iView of the navigation panel.
- **Dynamic Navigation:** The content of the related iViews and pages are displayed in the Dynamic Navigation iView of the navigation panel.
- **Target Components:** Links to the related iViews and pages are displayed in the Drag&Relate Targets iView of the navigation panel.

Procedure

1. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

2. Perform a lookup of the iView to which you want to add related items, and cast the object as an `IiView` object.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);
    IiView myIView =(IiView)iCtx.lookup(iViewId);

...
```

3. Create a descriptor (`INewObjectDescriptor` object) for the iView or page that you want to add as a related link.

```
    INewObjectDescriptor iViewDescriptor =
        (INewObjectDescriptor)iViewSrv.instantiateDescriptor
            (CreateMethod.DELTA_LINK,
                "pcd:portal_content/testxml", request.getUser());
```

4. Add the related item descriptor to the iView.

```
...

    myIView.addRelatedItem(iViewDescriptor,"testxml",
        RelatedItemType.DYNAMIC_NAVIGATION);

}
catch(NamingException e)
{
}
```

Pages

This section describes the tasks that can be performed for page semantic objects

- Adding iViews to a Page [Page 25]
- Removing iViews from a Page [Page 26]
- Adding Layouts to a Page [Page 27]
- Setting the Default Layout for a Page [Page 28]

For information on creating, modifying and deleting pages and other semantic objects, see All Semantic Objects [Page 16].

Adding iViews to a Page

This section describes how to add an iView (or page) to a page.

Procedure

1. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

2. Perform a lookup of the page to which you want to add an iView.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);
    IPage myPage =(IPage)iCtx.lookup(
        "pcd:portal_content/Desktop/finance");

...
```

3. Create a descriptor (`INewObjectDescriptor` object) for the iView or page that you want to add to your page.

```
    INewObjectDescriptor iViewDescriptor =
        (INewObjectDescriptor)iViewSrv.instantiateDescriptor
            (CreateMethod.DELTA_LINK,
                "pcd:portal_content/testxml", request.getUser());
```

4. Add the iView descriptor to the page.

```
    myPage.addiView(iViewDescriptor,"testxml");
```

The iView is automatically displayed at the bottom of the left-most column of the layout that is currently being used for the page.

If you want to place the iView into a particular column of a particular layout, you can specify a layout container into which to add the iView. The following adds the iView into the `com.sap.portal.reserved.layout.Cont2` container (second column) of the current layout:

```
...

    myPage.addiView(IVtoAdd,"testxml",
        "com.sap.portal.reserved.layout.Cont2");
}
catch(NamingException e)
{
}
```

`com.sap.portal.reserved.layout.Cont2` is the container ID of the second column as defined in a standard layout.

If an iView with the same atomic name exists in the page, an error is thrown.

Removing iViews from a Page

This section describes how to remove an iView from a page.

Procedure

1. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

2. Perform a lookup of the page from which you want to remove an iView.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);
    IPage myPage =(IPage)iCtx.lookup(
        "pcd:portal_content/Desktop/finance");

...
```

3. Remove the iView from the page. Use the atomic name of the iView.

```
...

   myPage.removeiView("myIView");

}
catch(NamingException e)
{
}
```

Adding Layouts to a Page

This section describes how to add a layout to a page.

Each page is assigned a default layout, which is used for rendering the page. You can assign other layouts to a page to enable users to personalize the page. Users can then select one of the other layouts assigned to the page.

Procedure

1. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

2. Perform a lookup of the page to which you want to add an iView.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);
    IPage myPage =(IPage)iCtx.lookup(
        "pcd:portal_content/Desktop/finance");

...
```

3. Create a descriptor (`INewObjectDescriptor` object) for the layout that you want to add to your page.

```
    INewObjectDescriptor layoutToAdd =
        (INewObjectDescriptor)iViewSrv.instantiateDescriptor
            (CreateMethod.DELTA_LINK,
                "pcd:portal_content/templates/layouts/narrowWide",
                    request.getUser());
```

4. Add the layout descriptor to the page.

```
...

    myPage.addLayout(layoutToAdd, "newLayout");
}
catch(NamingException e)
{
}
```

If a layout with the same atomic name exists in the page, an error is thrown.

Setting the Default Layout for a Page

This section describes how to set a page's default layout. A page's default layout is the layout used for rendering if the page's layout has not been personalized.

Procedure

1. Set the parameters for the JNDI lookup.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

2. Perform a lookup of the page for which you want to set the default layout.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);
    IPage myPage =(IPage)iCtx.lookup(
        "pcd:portal_content/Desktop/finance");

...
```

3. Set the default layout by specifying the layout's atomic name.

```
    myPage.setActiveLayout("narrowWideNarrow");
```

4. Save the changes.

```
...

    myPage.save();

}
catch(NamingException e)
{
}
```

Layouts

This section describes the tasks that can be performed for layout semantic objects:

- Adding iViews to a Page (via Layout) [Page 30]: You can specify where within a container to place an iView that has already been added to the page.

The `ILayout` interface also enables you to get information about a layout and its containers, with the help of the following methods:

- **`getContainerID()`:** Returns the ID for a specific container in the layout. Specify the container by its name as defined in the `portalapp.xml`.
- **`getContaineriViews()`:** Returns an array of strings that represent the atomic names of the iViews and pages within a specific container. Specify the container by its ID.
- **`getContainerIDs()`:** Returns an array of strings that represents the IDs of the containers in the layout.

For information on creating, modifying and deleting layouts and other semantic objects, see All Semantic Objects [Page 16].

Adding iViews to a Page (via Layout)

This section describes how to position an iView within a container of a layout on a page.

Procedure

1. Set the parameters for a JNDI lookup in the PCD.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT, PcmConstants.ASPECT_SEMANTICS);
```

2. Perform a lookup of the page on which you want to position an iView.

```
InitialContext iCtx = null;
try
{
    iCtx = new InitialContext(env);
    IPage myPage =(IPage)iCtx.lookup(
        "pcd:portal_content/Desktop/finance");
```

3. Get a reference to the layout in which you want to move the iView, for example, by getting the default layout.

```
    ILayout myLayout = myPage.getActiveLayoutObject();
```

   You can also get a list of a page's layouts by calling `getLayouts()` on the page.

4. Position the iView by calling `setiViewContainer()` and specifying the iView that you want to move, a container, and the position within the container to which you want to move the iView. The first position is `0`, the second position is `1`, and so forth.

```
    myLayout.setiViewInContainer("testxml",
        "com.sap.portal.reserved.layout.Cont1",1);
```

   This moves the `testxml` iView in the page to the second position in the `com.sap.portal.reserved.layout.Cont1` container of the default layout.

5. Save the changes to the layout.

```
...

    myLayout.save();

}
catch(NamingException e)
{
}
```

Systems

This section describes the tasks that can be performed for system semantic objects:

- Getting/Setting System Aliases [Page 32]
- Getting User Mapping [Page 33]
- Getting Aliases for All Systems [Page 34]

Systems are sets of properties that represent an external back-end application. Systems, such as for JDBC-compliant databases or SAP systems, enable connections to specific applications of these types and the retrieval of data.

For more information on systems, see System Landscape [External].

For information on creating, modifying and deleting systems and other semantic objects, see All Semantic Objects [Page 16].

Getting/Setting System Aliases

This section describes how to get and set system aliases.

The examples below use `mySystem`, which is an `ISystem` object.

- Get a system's aliases.

```
String[] aliases = mySystem.getAliases();
```

- Add/remove an alias for a system.

```
mySystem.addAlias("alias2");

mySystem.removeAlias("alias1");
```

- Set a system's default alias.

```
mySystem.changeDefaultAlias ("alias2");
```

Getting User Mapping

This section describes how to retrieve the user mapping (that is, the user name and password) associated with a system for the current user. The portal tries to connect to the back-end system using this user name and password when the current user requests an iView that connects with the system.

The example below uses `mySystem`, which is an `ISystem` object.

Procedure

1. Get the `ISystemUserMappingData` object associated with the system for the current user.

```
ISystemUserMappingData userMappingData = mySystem.getUserMappingData(
    request.getUser());
```

2. Get the user name or password from the `ISystemUserMappingData` object.

```
userMappingData.getUser();
userMappingData.getPassword();
```

Getting Aliases for All Systems

The `ISystems` interface provides methods for retrieving aliases for all systems in the PCD.

The examples below use `systems`, which is an `ISystems` object that can be obtained as follows:

```
ISystems systems = (ISystems)PortalRuntime.getRuntimeResources()
   .getService(ISystems.KEY);
```

- Get all aliases.

```
String[] aliases = systems.getAliases();
```

- Get all default aliases.

```
String[] defaultAliases = systems.getDefaultAliases();
```

Working with Administration (PCM) Objects

When working with PCD objects, you can use administration objects, which provide general methods that are valid for all semantic objects – less specific than methods exposed by semantic interfaces but semantically richer than those exposed by low-level PCD interfaces.

For more information on PCM and the purpose of administration interfaces, see Portal Content Model (PCM) [Page 14].

There is one main administration interface, `IAdminBase`, which you obtain directly from a PCD lookup, as described below. From the `IAdminBase` interface, you can derive the following additional interfaces:

- **`IAttributeSet`:** Provides access to an object's attributes, and enables you to look up, create, modify and delete attributes and meta-attributes, as described in Attributes [Page 37].
- **`IPermission`:** Provides access to an object's permissions, and enables you to check if a user has a specific permission for an object, as described in Permissions [Page 39].
- **`ICatalogNode`:** Provides information about the object that is useful for building an administration editor, as described in Catalog Node Attributes [Page 40].

  For example, this interface provides the icon and display name for the underlying object, which can then be displayed in an editor.

You can also find out the underlying object type of a PCD object by calling the `IAdminBase.getObjectType()` method.

Getting the IAdminBase Interface

The following describes how to get the main administration interface, `IAdminBase`, for a specific object.

1. Set the parameters for a JNDI lookup in the PCD, including setting the aspect to `ASPECT_ADMINISTRATION`.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT,
PcmConstants.ASPECT_ADMINISTRATION);
```

2. Perform a lookup by doing the following:
   a. Create an initial context with the parameters in your `Hashtable` object.
   b. Perform a lookup on your initial context, supplying the PCD name of the object, and then cast the returned object to `IAdminBase`.

```
InitialContext iCtx = null;
try {
    String objectName = "pcd:portal_content/myFolder/stocks";

    iCtx = new InitialContext(env);
    IAdminBase result =(IAdminBase)iCtx.lookup(objectName);
}
catch(Exception e) {}
```

Getting Other Interfaces

The following shows how to get secondary administration interfaces (`IAttributeSet`, `IPermission` and `ICatalogNode`) from an `IAdminBase` object:

```
IAdminBase result =(IAdminBase)iCtx.lookup(objectName);

IAttributeSet myIview = (IAttributeSet)
    result.getImplementation(IAdminBase.ATTRIBUTE_SET);
```

Attributes

This section describes how get and set attributes using the `IAttributeSet` interface, which provides the following benefits over working with attributes with PCD APIs:

- `IAttributeSet` returns all attributes associated with the current object, not just those stored in the PCD.

  For example, the implementation of `IAttributeSet` for an iView gathers attributes and values for the current object from the following sources:

  - PCD, which stores a small set of basic attributes, as well as any changes made by administrators to any attribute
  - Component from which the iView is derived, for example, the properties defined in the `portalapp.xml` for a portal component.
  - Core iView, located at *Portal Content* → *Content Provided by SAP* → *Core Objects* → *Core iView*. All iViews inherit from this iView.

  The priority is in the order shown. If an attribute is defined in the PCD and in the portalapp.xml, the PCD value takes precedence.

- The implementation of `IAttributeSet` can enforce rules on modifying attributes, whether general rules for all semantic objects or for the current semantic object.

  For example, `IAttributeSet` can restrict you from modifying the `Merge Priority` property if the `Can Be Merged` property is **`false`**.

For information on how to obtain an `IAttributeSet` object for a PCD object, see Working with Administration Objects [Page 35].

For information on working with attributes with PCD APIs, see Contexts and Attributes [Page 43].

Getting/Setting Attributes

The following gets the `Object is a Template` attribute, and its `Inheritance` meta-attribute:

```
IAdminBase myAdmin = (IAdminBase)iCtx.lookup(myObject);
IAttributeSet attrSet = (IAttributeSet)
   myAdmin.getImplementation(IAdminBase.ATTRIBUTE_SET);


// Display Object is a Template attribute
response.write(attrSet.getAttribute(
    IAttrPcmGeneral.ATTRIBUTE_IS_TEMPLATE) );

// Display Inheritance meta-attribute of Object is a Template
attribute
response.write(attrSet.getMetaAttribute(
    IAttrPcmGeneral.ATTRIBUTE_IS_TEMPLATE,
        IAttrPcmGeneral.META_ATTRIBUTE_INHERITANCE));
```

The following sets the `Object is a Template` attribute to **`true`**:

```
attrSet.putAttribute(IAttrPcmGeneral.ATTRIBUTE_IS_TEMPLATE,true);
attrSet.save();
```

If the attribute does not exist, it is created.

Text Attributes

Text attributes are translatable and, therefore, generally require an additional parameter that indicates the locale.

The following attributes require the use of the locale:

- `com.sap.portal.pcm.Title`
- `com.sap.portal.pcm.Description`

The following meta-attributes require the use of the locale:

- `plainDescription`
- `longDescription`
- `category`
- `validValueTitle0`, `validValueTitle1`, and so forth.

The following gets the `Title` text attribute:

```
response.write (attrSet.getAttribute(
    IAttrPcmGeneral.ATTRIBUTE_TITLE,request.getLocale()));
```

Attribute Constants

To specify an attribute, use the designated constant for that attribute. The constants for each semantic type are located in a corresponding interface in the `com.sap.portal.pcm.attributes` package.

For example, the constants for iView attributes are located in the `IAttriView` interface. The following code checks whether the current iView allows browser caching:

```
response.write(
    attrSet.getAttribute(IAttriView.ATTRIBUTE_ALLOW_BROWSER));
```

Permissions

With the `IPermission` interface, you can do the following:

- Check if a specific user has a specific permission – `isAllowed()`.
- Get a list of all the permissions that can be assigned for this object – `getAllPermissions()`.

The interface also has the helper function `getPermissionTitle()` for getting the display name for a specific permission.

In order to modify permissions, use the PCD API, as described in Permissions [Page 46].

Checking a User's Permission

The following code example checks whether the current user has read permission on the object with the PCD name stored in the string `myObject`:

```
IAdminBase myAdmin = (IAdminBase)iCtx.lookup(myObject);
IPermission myIview = (IPermission)
    myAdmin.getImplementation(IAdminBase.PERMISSION);

boolean hasPermission = myIview
    .isAllowed(request.getUser(),IPermission.PCM_ADMIN_READ);

if (hasPermission) {
    response.write(request.getUser().getName() + " has "
        + myIview.getPermissionTitle("Pcd.Read",request.getLocale())
            + " permission.<br>");
}
```

Permission Constants

The `IPermission` interface provides constants for checking the following permissions:

- `Owner`
- `Read`
- `Write`
- `Read-Write`
- `Full Control`
- `Use`

To check the role assigner permission, use the constant `IPortalBasicRoleFactoryService.ROLE_ASSIGNMENT_PERMISSION`.

The PCM's `IPermission` and PCD's `IPcdStandarPermissions` interfaces define different sets of permissions constants. They are not interchangeable.

Catalog Node Attributes

With the `ICatalogNode` interface, you can get information about portal objects in order to display them in administration editors or other applications. The interface, for example, is used by the Portal Catalog user interface to display the tree of portal content.

The following are the key methods:

- **`getTitle()`:** Returns the friendly name of the object.
- **`getIcon()`:** Returns a `PcmUri` object that enables you to get the URL for the image that represents the object's type.

The following retrieves the catalog node attributes for a portal object:

```
Hashtable env1 = new Hashtable();

env1.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env1.put(Context.SECURITY_PRINCIPAL, request.getUser());
env1.put(Constants.REQUESTED_ASPECT,
PcmConstants.ASPECT_ADMINISTRATION);

InitialContext iCtx = null;

try {
    iCtx1 = new InitialContext(env1);
    IAdminBase adminBase = (IAdminBase)iCtx.lookup(lookupObject);

    // Write type of portal object
    response.write("Object Type: " +
adminBase.getObjectType()+"<BR>");

    // Get ICatalogNode interface
    ICatalogNode cn = (ICatalogNode) adminBase
        .getImplementation(IAdminBase.CATALOG_NODE);

    // Write title of portal object
    response.write(cn.getTitle(request.getLocale())+"<BR>");

    // Get object for obtaining image URL
    PcmUri pcmUri = cn.getIcon(ObjectState.DEFAULT);

} catch (NamingException e) {}
```

Displaying Icons

The `PcmUri` is a generic object enabling you to derive the URL for an image. It is independent of the user interface technology.

The following code shows how to derive the URL from within a portal (PRT) application:

```
String iconURL = "";

ICatalogNode cn = (ICatalogNode) adminBase
    .getImplementation(IAdminBase.CATALOG_NODE);
PcmUri pcmUri = cn.getIcon(ObjectState.DEFAULT);

switch (pcmUri.getUrlType()) {

    case PcmUri.URL_TYPE_APPLICATION_RESOURCE:
        iconURL = request.getWebResourcePath(pcmUri.getApplication())
            + "/" + pcmUri.toString();
        break;

    case PcmUri.URL_TYPE_JNDI_URL:
        IPortalComponentURI componentURI =
            request.createPortalComponentURI();
        componentURI.setContextName(pcmUri.toString());
        iconURL = componentURI.toString();
        break;

    case PcmUri.URL_TYPE_ABSOLUTE:
        iconURL = pcmUri.toString();
        break;
}
```

The following code shows how to derive the URL from within a Web Dynpro application:

```
String iconURL = "";

ICatalogNode cn = (ICatalogNode) adminBase
    .getImplementation(IAdminBase.CATALOG_NODE);
PcmUri pcmUri = cn.getIcon(ObjectState.DEFAULT);

switch (pcmUri.getUrlType()) {

    case PcmUri.URL_TYPE_APPLICATION_RESOURCE:
        iconURL = WDPortalUtils
            .getPortalWebResourceURL(pcmUri.getApplication())
                + "/" + pcmUri.toString();
        break;

    case PcmUri.URL_TYPE_JNDI_URL:
        iconURL = pcmUri.toString();
        break;

    case PcmUri.URL_TYPE_ABSOLUTE:
        iconURL = pcmUri.toString();
        break;
}
```

Working with PCD Objects

The PCD provides low-level classes and interfaces in order to access the PCD, and to perform such tasks as creating contexts and attributes, changing permissions and deleting objects.

Whenever possible, developers should use the higher-level interfaces (semantic and administration objects), as described in Working with Semantic Objects [Page 15] and Working with Administration Objects [Page 35].

This section describes how to work with the following:

- Contexts and Attributes [Page 43]
- Delta Links [Page 44]
- Permissions [Page 46]
- Personalization [Page 52]

Contexts and Attributes

Using the standard JNDI APIs, you can add and remove contexts and modify context attributes.

However, when working with portal objects, it is easier to use the semantic or administration objects, as described in Working with Semantic Objects [Page 15] and Working with Administration (PCM) Objects [Page 35], or to create delta links with the PCD API, as described in Delta Links [Page 44].

Storing Data in the PCD

Using the standard JNDI APIs, you can create contexts and add attributes in order to store information in the PCD unrelated to the portal objects that administrators work with in the Portal Catalog.

It is recommended not to store large amounts of data in the PCD, as this can significantly reduce performance.

The following is an example of creating a top-level context (not under `portal_content`), and adding a multi-value attribute to the context:

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT,
IPcdAttribute.PERSISTENCY_ASPECT);

InitialContext   iCtx = null;

try {
    iCtx = new InitialContext(env);

    // Get the top-level context
    IPcdContext myPcdContext = (IPcdContext) iCtx.lookup("");

    // Create an attribute and set its value to {"Value1","Value2"}
    Attribute myAttr = new BasicAttribute("myAttribute","Value1");
    myAttr.add("Value2");
    Attributes myAttrs = new BasicAttributes();
    myAttrs.put(myAttr);

    // Create a subcontext with the predefined attributes
    myPcdContext.createSubcontext("myContext",myAttrs);
} catch (NamingException e) {}
```

Delta Links

The PCD provides delta link information about objects – such as whether the object is a delta link, what attributes were changed, and what attributes were erased.

Most information is derived from an `IDlModificationState` object that you can derive from the `IPcdContext` interface, as follows:

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
         IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT,
IPcdAttribute.PERSISTENCY_ASPECT);

InitialContext   iCtx = null;

String lookupObject = "portal_content/myFolder/myObject";

try {
    iCtx = new InitialContext(env);

    IPcdContext myPcdContext =(IPcdContext)
iCtx.lookup(lookupObject);

    IDlModificationState myDLState = myPcdContext
        .getDlModificationState("");
}
```

If the object is not a delta link or inherited via a delta link, `getDlModificationState()` returns null.

If an attribute's inheritance meta-attribute is set to `FINAL`, the attribute cannot be modified in a delta link.

Getting Delta Link Information

From an `IDlModificationState` object, you can perform the following tasks:

- Get the source object of a delta link.

```
response.write(myDLState.getSourceUrl());
```

- Get all attributes that were changed in a delta link.

```
Iterator myMods = myDLState.getModifiedAttributeIds();
while (myMods.hasNext()) {
    response.write("ID: " + myMods.next() + "<BR>");
}
```

- Get the changes made to a specific attribute of a delta link.

```
ModificationItem[] myMod =
    myDLState.getModifications("ForcedRequestLanguage");

for (int i=0;i<myMod.length;i++){
    response.write(myMod[i].getAttribute().getID()+"<BR>");
    response.write(myMod[i].getModificationOp()+"<BR>");
}
```

The `getModificationOp()` method returns one of the following constants:

- `DirContext.ADD_ATTRIBUTE`
- `DirContext.REMOVE_ATTRIBUTE`
- `DirContext.REPLACE_ATTRIBUTE`

For each attribute, a delta link stores all the changes made to the attribute, not just the final result.

Creating a Delta Link

The following creates a delta link:

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Constants.REQUESTED_ASPECT,
IPcdAttribute.PERSISTENCY_ASPECT);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());

InitialContext iCtx = null;
String myFolderName = "pcd:portal_content/myFolderB";

try {
    iCtx = new InitialContext(env);

    IPcdContext myFolderB = (IPcdContext) iCtx.lookup(myFolderName);

    myFolderB.createDeltaLink(
        "myDL",null,"portal_content/myFolderA/myIView");
}
catch (Exception e) {}
```

The above creates a PCD object called `myDL` in `myFolderB`, which is a delta link to `myIView` in `myFolderA`.

Resetting Attributes

The following resets all attributes changed in a delta link object:

```
myPcdContext.removeModifications("");
```

The following resets a specific attribute (`ForcedRequestLanguage`) that was changed in a delta link object:

```
String myAttrsToDelete[] = {"ForcedRequestLanguage"};
myPcdContext.removeAttributeModifications("",myAttrsToDelete);
```

Permissions

Access to PCD content is protected by access control lists (ACLs). Each object has an ACL, either specific for that object or inherited from its parent.

An ACL is a set of access control entries (ACEs), each of which specifies a principal (that is, user, group or role) and the permission granted to that principal.

The permission is specified by a string, and any string can be specified. Developers can define their own permissions and store them in the PCD. The PCD uses its own set of permission strings (defined in `IPcdStandardPermissions`) to determine user rights for PCD operations on each PCD object.

Only unit objects and folders have ACLs.

Inheritance Rules

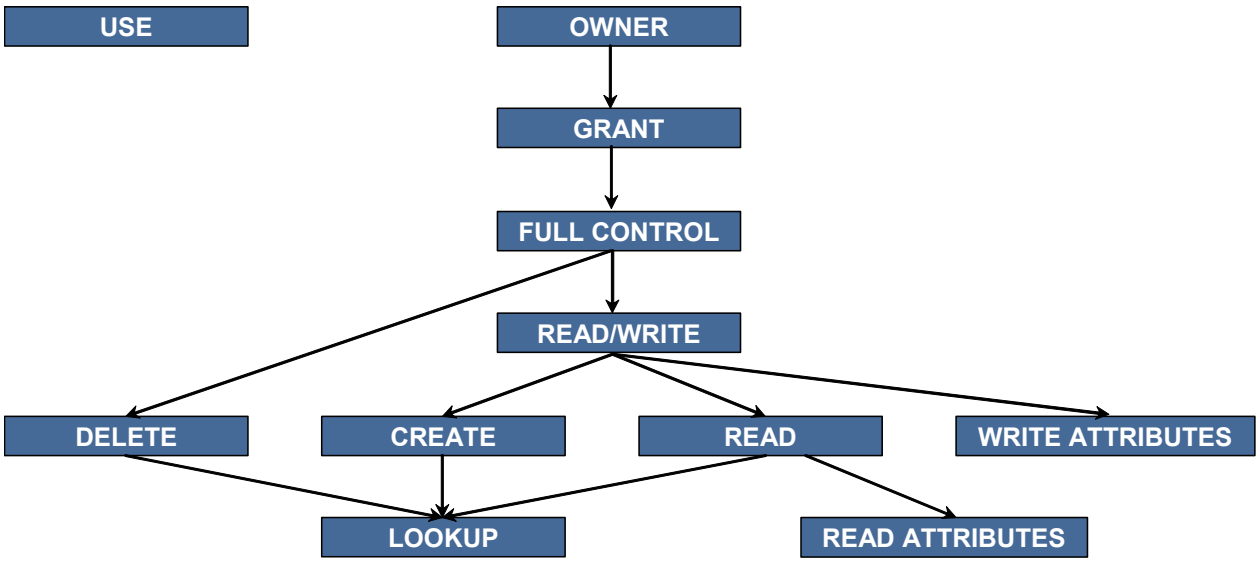
The following are inheritance rules for ACLs:

- An object inherits permissions from its parents if it has no ACL of its own.
- If an object has an ACL, the new ACL replaces the parents ACL and the inheritance is broken.
- When accessing a PCD object, only the object's permissions are checked. Permissions of objects in the lookup path are not checked.

Types of Permissions

The operations that a user can perform on an object are based on the permissions that the user is granted for that object.

Each permission entitles a user to a specific set of operations on the current object, including all operations permitted by its children permissions. For example, the FULL CONTROL permission entitles the user to perform a set of operations, which includes those permitted by the DELETE and the READ/WRITE permissions.



Refer to specific permissions with the constants defined by `IPcdStandardPermissions`.

Allowed Operations

The following is a list permissions and the PCD operations – which are generally performed on `IPcdContext`, `IDeltaLink` or `IAclHandle` objects – that each permission enables:

Permission	Operations/Methods
OWNER	Includes all permissions for GRANT.

GRANT	`IAclHandle.createAcl()`
	`IAclHandle.removeAcl()`
	`IAcl.createAclEntry()`
	`IAcl.removeAclEntry()`
	`IAcl.removeOwner()`
	This permission provides the same operations as OWNER. It is recommended to use OWNER instead.

FULL CONTROL	Includes all permissions for DELETE and READ/WRITE.
READ/WRITE	Includes all permissions for CREATE, READ and WRITE ATTRIBUTES.

DELETE	`IPcdContext.destroySubcontext()`
	`IPcdContext.unbind()`
	The above operations can be performed on a unit object. For subobjects of a unit, only WRITE ATTRIBUTES is required.

CREATE	`IPcdContext.createSubcontext()`
	`IPcdContext.bind()`
	`IPcdContext.rebind()`
	`IPcdContext.createDeltalink()`

READ	Includes all permissions for LOOKUP and READ ATTRIBUTES.

WRITE ATTRIBUTES	`IPcdContext.modifyAttributes()`
	For objects that are subobjects of a unit, the following can also be performed:
	`IPcdContext.destroySubcontext()`
	`IPcdContext.unbind()`

READ ATTRIBUTES	`IPcdContext.getAttributes()`

LOOKUP	`IPcdContext.lookup()`
	`IPcdContext.lookupLink()`
	`IPcdContext.list()`
	`IPcdContext.listBindings()`
	`IPcdContext.search()`

USE	Any of the above operations if the PCD is called in personalization mode (except for those defined by the GRANT permission). For information on personalization, see Personalization [Page 52].

Looking Up/Adding/Removing Permissions

The following are code samples for working with permissions.

Looking Up Permissions

The following displays all the ACEs for the object `portal_content/myFolder/myObject`.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT,
IPcdAttribute.PERSISTENCY_ASPECT);

InitialContext iCtx = null;

String lookupObject = "portal_content/myFolder/myObject";

try {

    iCtx = new InitialContext(env);
    IPcdContext myPcdContext =(IPcdContext)
iCtx.lookup(lookupObject);

    IAclHandle myAclHandle = myPcdContext.getAclHandle();

    // Get ACL for this object
    IPermissionCheckAcl thePerms =
myAclHandle.getAclForPermissionCheck();

    // Get ACEs for this object
    Iterator myIt = thePerms.getAclEntries().iterator();

    while (myIt.hasNext()) {

        // Get next ACE
        IAclEntry ace = (IAclEntry) myIt.next();

        IPrincipal myPrincipal = (IPrincipal) ace.getPrincipal();

        // Display principal name and permission
        response.write(myPrincipal.getDisplayName() + "--" +
            ace.getPermission() + "<BR>");
    }
}
```

Adding Permissions

The following adds the READ/WRITE permission for the user `myUser` for the object `portal_content/myFolder/myObject`.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT,
IPcdAttribute.PERSISTENCY_ASPECT);

InitialContext iCtx = null;

String lookupObject = "portal_content/myFolder/myObject";

// Create user object to which to add permission
IUserFactory userFactory = UMFactory.getUserFactory();
IUser myUser = null;

try {
    myUser = userFactory.getUserByLogonID("myUser");

    iCtx = new InitialContext(env);
    IPcdContext myPcdContext =(IPcdContext)
iCtx.lookup(lookupObject);

    IAclHandle myAclHandle = myPcdContext.getAclHandle();

    // Add permission to the IAcl object for this PCD object
    myAclHandle.getOwnAcl().createAclEntry(
        request.getUser(),myUser,
            IPcdStandardPermissions.PCD_PERMISSION_READ_WRITE);
}
```

Removing Permissions

The following removes the USE permission for the user `myUser` for the object `portal_content/myFolder/myObject`.

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT,
IPcdAttribute.PERSISTENCY_ASPECT);

InitialContext iCtx = null;

String lookupObject = "portal_content/myFolder/myObject";

IUserFactory userFactory = UMFactory.getUserFactory();
IUser myUser = null;

try {
    // Create user object for which we want to remove permission
    myUser = userFactory.getUserByLogonID("myUser");

    // Look up object
    iCtx = new InitialContext(env);
    IPcdContext myPcdContext =(IPcdContext)
iCtx.lookup(lookupObject);

    // Get ACL handle
    IAclHandle myAclHandle = myPcdContext.getAclHandle();

    // Get ACL
    IAcl thePerms = myAclHandle.getOwnAcl();

    // Get ACEs for specific user
    Iterator myIt = thePerms.getAclEntries(myUser).iterator();

    while (myIt.hasNext()) {

        // Get next ACE
        IAclEntry ace = (IAclEntry) myIt.next();

        // Remove ACE if it is for USE permissions
        if (ace.getPermission().equals(
            IPcdStandardPermissions.PCD_PERMISSION_USE)){

            thePerms.removeAclEntry(request.getUser(),ace);
        }
   }
}
```

Personalization

The PCD can create different views of the PCD for different users.

In other words, for each attribute of an object, the PCD stores a default value and can also store different values for each user. When the PCD is queried for a specific user, the personalized value for that user is returned instead of the default value.

For example, an iView that displays the weather can have an attribute called `City`. Each user can set the `City` attribute to a different city, so that the weather for that city is displayed when that user displays the iViews.

If an attribute's inheritance meta-attribute is set to FINAL, the attribute cannot be personalized.

Getting a Personalized View of the PCD

You can obtain the personalized view of the PCD for a specific user by supplying the `IPcdContext.PCD_PERSONALIZATION_PRINCIPAL` variable with an `IUser` object for the user when creating the initial context, as follows:

```
// Create user for personalization
IUserFactory userFactory = UMFactory.getUserFactory();
IUser myUser = null;

myUser = userFactory.getUserByLogonID("userName");

// Create environment variables for initial context
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(Constants.REQUESTED_ASPECT,
IPcdAttribute.PERSISTENCY_ASPECT);

// Personalization variable
env.put(IPcdContext.PCD_PERSONALIZATION_PRINCIPAL,
request.getUser());
```

All PCD operations on an object will only affect the personalized view of that object.

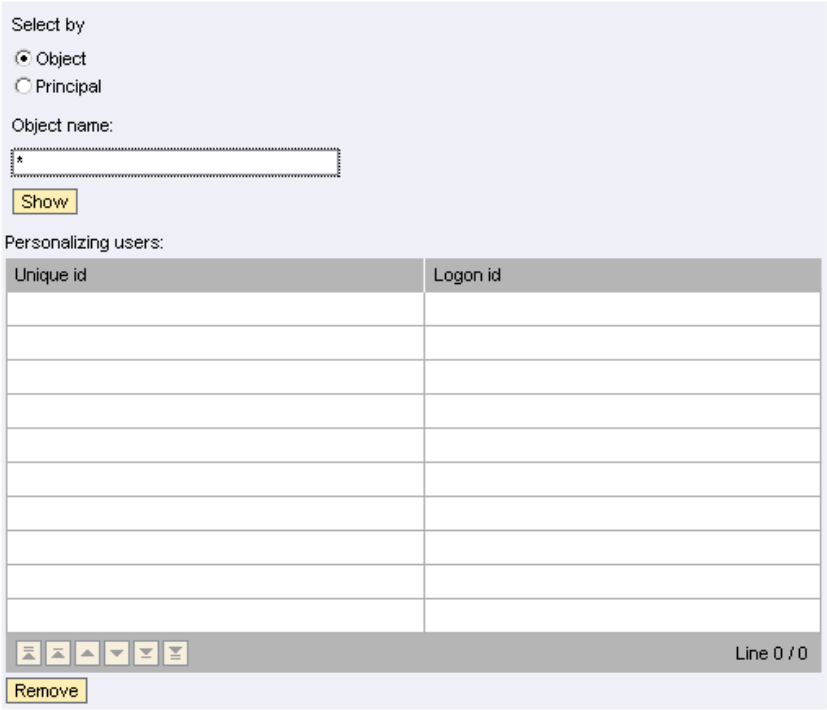
Removing Personalization

The portal provides a tool called Personalization Cleanup that enables an administrator to remove a specific user's personalization on a specific object.

This is particular useful when an attribute is defined in the `portalapp.xml` of a portal component, then is personalized, and then needs to be updated in the `portalapp.xml`. After the new portal component is deployed, the personalized value is still present in the PCD, and supercedes even the new value in the `portalapp.xml`.

Running Personalization Cleanup

To run the tool, go to *System Administration* → *Support* → *Portal Content Directory* → *Personalization Cleanup.*



To remove personalization for a specific object, do one of the following:

- **Search by Object**
  a. Select the *Object* radio button.
  b. Enter the PCD name of the object (without the `pcd:` prefix)
  c. Click *Show*. All users with personalization for the specified object are displayed.
  d. Select the users whose personalization you want to remove.
  e. Click *Remove*.
- **Search by User, Group or Role (Principal)**
  a. Select the *Principal* radio button.
  b. Enter a search string for the user, group or role whose personalization you want to remove.
  c. Click *Show*. All users who fit the search string and with any personalization in the PCD are displayed.

d. Select the logon ID of the user, group or role whose personalization you want to remove. All objects personalized by the selected principal are displayed.

e. Select the objects from which you want to remove personalization for the selected user, group or role.

f. Click *Remove*.

Removing Personalization via Code

Personalization can be removed by performing a personalized lookup (a lookup in which a personalization principal is specified), and then removing some or all of the modifications by calling `removeModifications()` or `removeAttributeModifications()` on the `IPcdContext` object. This removes the personalization for the user specified as the personalization principal.

These are the same methods for removing delta link modifications when performing a nonpersonalized lookup.

The following removes all personalization from an object for the current user:

```
Hashtable env = new Hashtable();

env.put(Context.INITIAL_CONTEXT_FACTORY,
    IPcdContext.PCD_INITIAL_CONTEXT_FACTORY);
env.put(Constants.REQUESTED_ASPECT,
IPcdAttribute.PERSISTENCY_ASPECT);
env.put(Context.SECURITY_PRINCIPAL, request.getUser());
env.put(IPcdContext.PCD_PERSONALIZATION_PRINCIPAL,
request.getUser());

InitialContext iCtx = null;

String myObjectName = "pcd:portal_content/myFolder/myObject";

try {
    iCtx = new InitialContext(env);

    IPcdContext myObject = (IPcdContext) iCtx.lookup(myObjectName);

    myObject.removeModifications("");
}
catch (Exception e) {}
```

.