Sun Java System Portal Server 7.1 Developer's Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819–5070–10
March 2007

070226@16599

Contents

Figures

Tables

Examples

Preface

The Sun™ Java™ System Portal Server 7.1 Developer's Guide provides a high-level overview of the Sun Java SystemPortal Server APIs. This book explains how to extend and customize the Portal Server APIs.

Who Should Use This Book

This *Developer's Guide* is intended for use by developers and individuals responsible for customizing the Portal Server APIs and creating custom providers and portlets for use with their deployment environment.

- Lightweight Directory Access Protocol (LDAP)
- Java technology
- JavaServer Pages™ (JSP) technology
- Hypertext Transfer Protocol (HTTP)
- Hypertext Markup Language (HTML)
- Extensible Markup Language (XML)

Also, this book assumes that you already know the basics of the Solaris™ Operating Environment and UNIX® command-line utilities and administrative tasks.

Related Books

- *Sun Java System Portal Server 7.1 Command Line Reference*
- *Sun Java System Portal Server 7.1 Technical Reference*

Related Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (`http://www.sun.com/documentation/`)
- Support (`http://www.sun.com/support/`)
- Training (`http://www.sun.com/training/`)

Typographic Conventions

The following table describes the typographic conventions that are used in this book.

TABLE P–1 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell	`machine_name%`
C shell for superuser	`machine_name#`
Bourne shell and Korn shell	`$`
Bourne shell and Korn shell for superuser	`#`

Default Paths and File Names

The following table describes the default paths and file names used in this book.

TABLE P–3 Default Paths and File Names

Term	Description
PortalServer-base	Represents the base installation directory for Sun Java System Portal Server 7.1 software. The Portal Server software default base installation and product directory depends on your specific platform: Solaris systems: `/opt/SUNWportal` Linux systems: `/opt/sun/portal`
AccessManager-base	Represents the base installation directory for Sun Java System Access Manager software. The Access Manager software default base installation and product directory depends on your specific platform: Solaris systems: `/opt/SUNWam`
DirectoryServer-base	Represents the base installation directory for Sun Java System Directory Server software. The Directory Server 5.2 software default base installation is `/var/opt/mps/serverroot`. For the DirectoryServer 6.0, the base location for Solaris is `/opt/SUNWdsee/ds6` and `/opt/sun/ds6`. The DirectoryServer data directory or instance location (in version 6.0), which is used by `dsadm` to start and stop instances is `/var/opt/SUNWdsee/dsins1`.
ApplicationServer-base	Represents the base installation directory for Sun Java System Application Server software. The Application Server software default base installation is `/opt/SUNWappserver`.

TABLE P–3 Default Paths and File Names *(Continued)*

Term	Description
WebServer-base	Represents the base installation directory for Sun Java System Web Server software. The Web Server 6.1 spx software default base installation is `/opt/SUNWwbsvr`. For WebServer 7.0, the default base location is `/opt/SUNWwbsvr7` for Solaris and `/opt/sun/SUNWwbsvr7` for Linux. The WebServer data directory or instance location, which is used to start and stop instances is `/var/opt/SUNWwbsvr7`.
PortalServer-DataDir	Represents the directory where JSPs, templates and property files, and tag libraries are installed. By default, this is: ■ `/var/opt/SUNWportal/` on Solaris ■ `/var/opt/sun/portal` on Linux

◆ ◆ ◆ CHAPTER 1

Introduction to Portal Server

This chapter provides an introduction to customizing the portal to fit the specific needs of your organization. It describes the Sun Java System Portal Server architecture, APIs, and programming concepts that developers can use.

This chapter contains the following sections:

Note – Detailed information on the Portal Server APIs is available in the Java docs. The URL to access the Java docs is:

`http://java.sun.com/javase/6/docs`

Overview of Portal Server

This section provides a brief description of the Portal Server. See *Sun Java System Portal Server 7.1 Technical Overview* for a complete product architecture description.

What Is a Portal?

A portal is a doorway or entry point to a set of resources that an enterprise wants to make available to the portal's users. For some consumer portals, the set of resources includes the entire World-Wide Web. For most enterprise portals, the set of resources includes information, applications, and other resources that are specific to the relationship between the user and the enterprise.

The primary purpose of the Sun Java System Portal Server is to give end users a portal Desktop, which provides access to resources and applications. In addition, a search engine infrastructure enables intranet content to be organized and accessed from the portal Desktop.

Portal Server Architecture

This section describes the Portal Server architecture. In Figure 1–1, the Portal Server components consist of:

- Services such as search (made up of the search engine and robot), discussions, and subscriptions
- Content delivery and presentation on the Desktop

The Java Development Kit™ (JDK) provides the Java run-time environment for all Java software in Portal Server and its underlying components.

The Sun Java System Web Server, or the Sun Java System Application Server, or the BEA WebLogic Server, or the IBM WebSphere Application Server provides the web container. The application server provides the Portal Server with a robust, highly scalable web container for the Portal Server web applications. It also provides the Portal Server with compatibility for other applications written to use application server and for developers that want to use the additional enterprise services that are provided by these application servers.

The Directory Server provides the user profile data repository. The Access Manager provides support for core services such as profile, session, authentication, and logging. It also provides single-sign-on services, policy management, debug utility, and client support interfaces for the Portal Server. Use the Access Manager administration console for user administration.

The Portal Server Search Engine provides the search capability. It includes basic and advanced search and browse channels for the Desktop. It uses a robot to create resource descriptions for documents that are available in the intranet, and stores these resource descriptions in an indexed database. Search includes Java and C APIs for submitting resource descriptions and for searching the database. Search also includes an administration console module for editing Search service data and for configuring the search engine and the robot. The console also lets you edit the contents of the Search database.

The Desktop provides the primary end-user interface for the Portal Server and a mechanism for extensible content aggregation through the Container Provider Interface (PAPI). The Desktop includes a variety of providers that provide a container hierarchy and the basic building blocks for building some types of channels. The Desktop implements a display profile data storage mechanism on top of an Access Manager service for storing content provider and channel data.

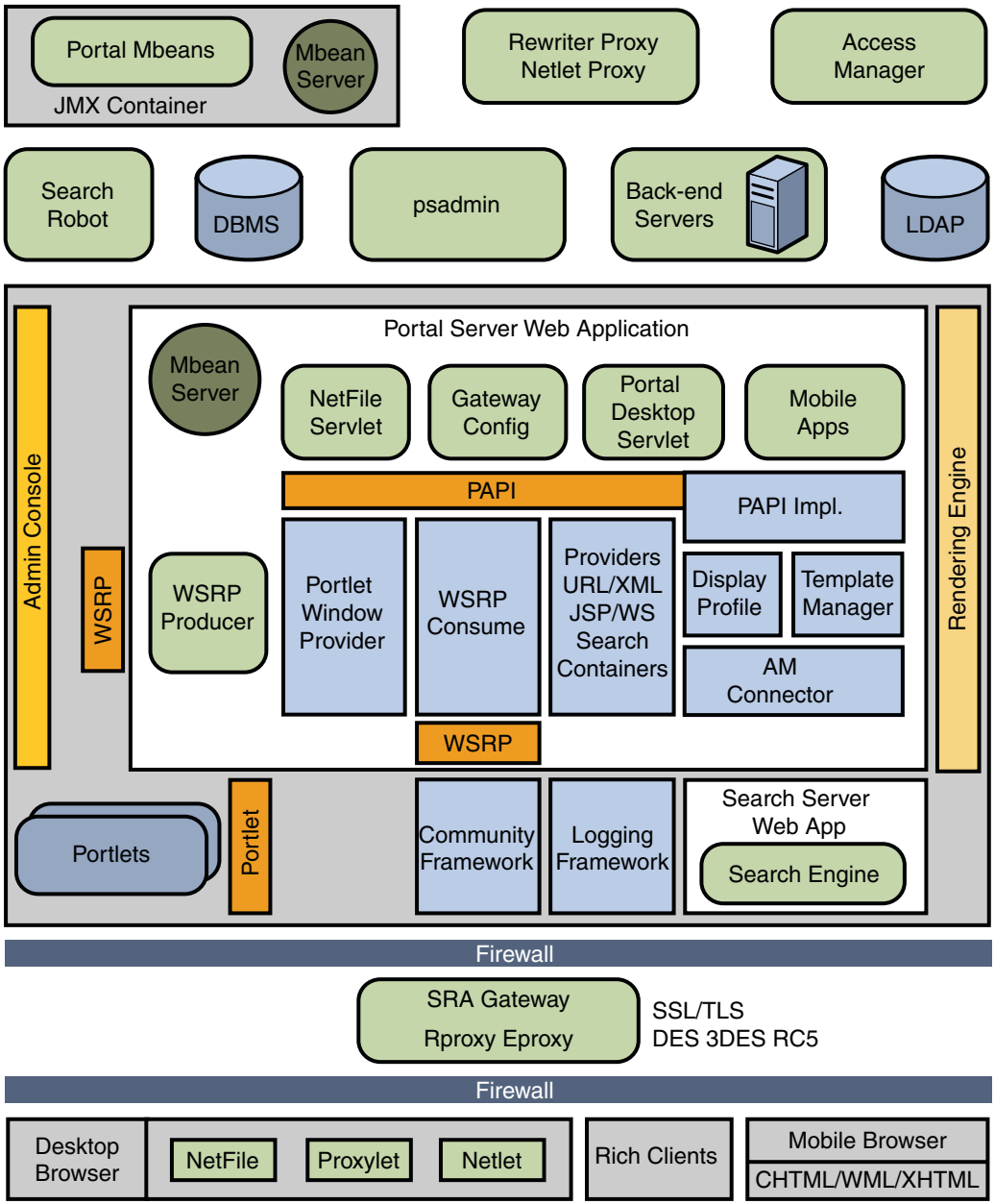

FIGURE 1–1 Portal Software Architecture

Portal Desktop

The Portal desktop is a logical component, which consists of the Desktop servlet, Provider API and SPI, various channels, and other support APIs and utilities.

Sample Desktop and Desktop Hierarchy

The sample desktops use an aggregation of a variety of separate web applications (channels) within a common framework. The common framework provides multiple levels and styles of aggregation, presented to end users through a container metaphor.

Providers and Channels

In the Desktop, leaf channels are the basic unit of content, displaying a specific type of information. To the end user, a channel is a distinct unit of content in the Desktop, usually (but not always) set off with a border and header row of icons that enables users to configure the channel to their preference. A provider is a Java class responsible for converting the content, from a file or an application or service, into a presentable format for a channel.

Portlets and Web Service for Remote Portlets

Portlets and Web Service for Remote Portlets (WSRP) are available with Portal Server. Portlets are web components specifically designed to be aggregated in the context of a composite page. Java Specification Request (JSR) 168 specifies how portlets interact, how their life cycles are managed, and provides details of their semantics.

WSRP is a specification that defines a web service interface to access and interact with interactive presentation-oriented web service. WSRP is a standard to access and get the content using the portlets deployed in remote server using web services.

Portal Server Software Desktop

The Desktop provides a mechanism for extending and aggregating content through the Provider Application Programming Interface (PAPI). The PAPI is a Java API that enables you to construct the basic building blocks for creating channels. Usually, though not always, channels contain content. You can also have channels of channels; that is, a container channel that aggregates other channels. A channel can also be the entire Desktop page. The container channels define the layout of the Desktop.

The figure below shows a simple representation of a portal Desktop and its providers and containers. In this figure, the Desktop front page is a tab container with two tabs. Each tab contains a table container with various channels.

Notice how one provider, in this case, URLScraperProvider, is serving more than one channel. Providers can have a one-to-many relationship with channels.

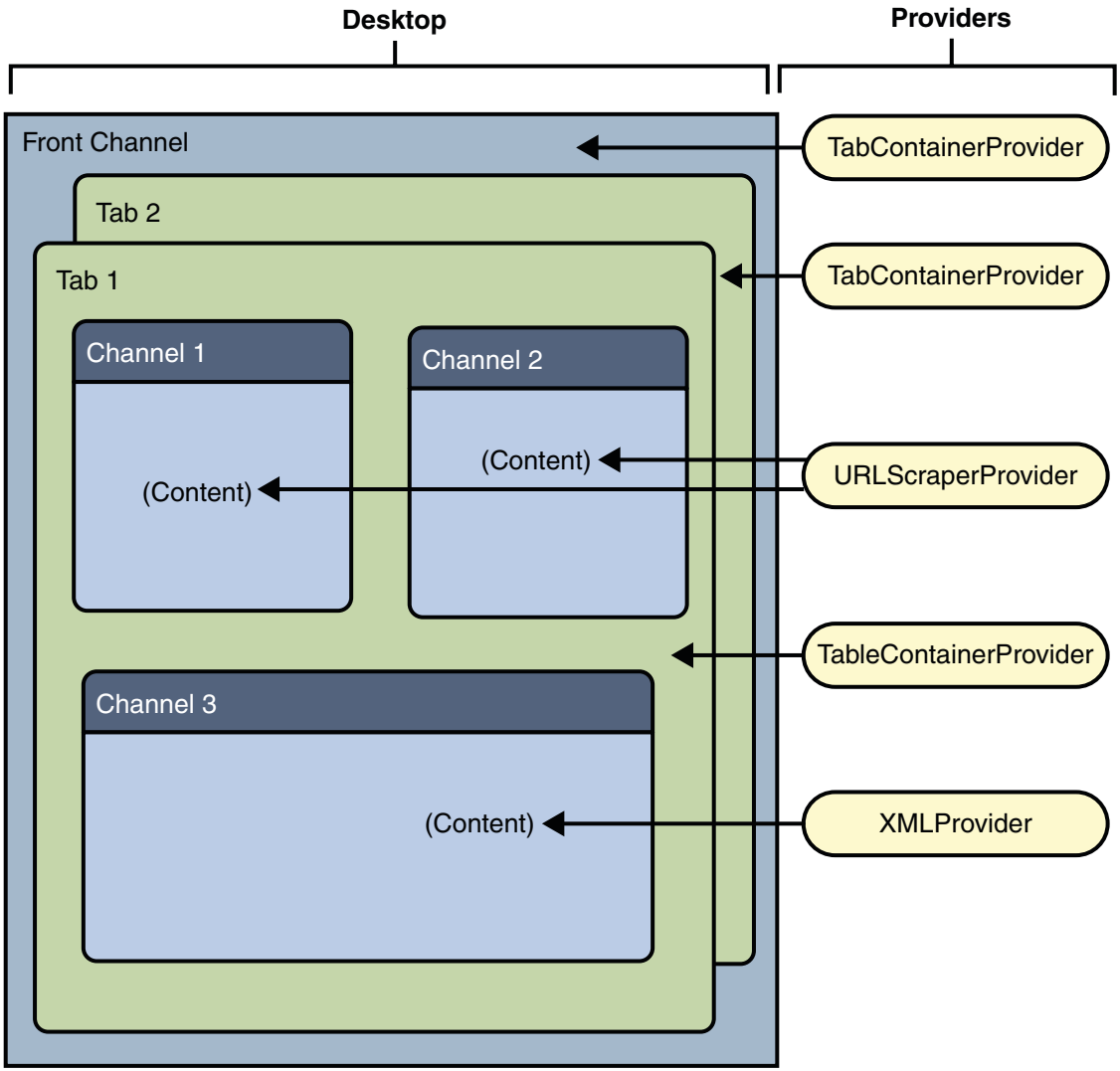

FIGURE 1–2 Desktop Hierarchy and Building Block Providers

Search Engine

The Portal Server Search Engine is a taxonomy and database service designed to support basic and advanced search and browse channels for the Desktop. The search engine uses a robot to create resource descriptions (RDs) for documents that are available in the intranet, and stores these resource descriptions in an indexed database. Resource descriptions can also be imported from another server or from a backup Search (Summary Object Interchange Format) file. The

search engine includes Java and C APIs for submitting resource descriptions and for searching the database. The search engine database can also be used for storing other arbitrary content such as a shared content cache for other content providers.

Access Manager Software Services

Portal applications and resources use Access Manager to provide services such as authentication, single-sign-on, profile, and session management. See the Access Manager documentation for more information.

Portal Environment

This section explains the following component products associated with the Portal Server:

Java Development Kit

The minimum JDK version required for Portal Server environment is 1.5.0_05 for Solaris SPARC® 9 and 10, Solaris x86 9 and 10, and Linux operating systems.

Web Container

The Portal Server providers run within the Java Virtual Machine provided by the web container, which may vary between different web containers. Support for web containers provided in this release are Sun Java System Web Server 7.0 and Sun Java System Application Server 8.2. It is also recommended that the support for Sun Java System Web Server 6.1 and Sun Java System Application Server 7.0 and 8.1 be provided in this release.

Note – Version 7.0 is not supported on HP-UX and also there are no Java enterprise System wide requirements to support third party web container technology.

Directory Server

The Portal Server environment supports the current and prior releases of the Sun Java System Directory Server 5.2 (R2–R4) and 6.0 (R5).

Note – Currently, there are no Java Enterprise System wide requirements to support third party LDAP servers.

File System

The Portal Server product uses `/opt/SUNWportal` (`/opt` is a default that can be changed during installation), `/etc/opt/SUNWportal`, and `/var/opt/SUNWportal` for installing Portal Server specific packages and other files into the file system.

Most Portal Server Java classes are defined in packages under the `com.sun.portal` package name.

The Portal Server is installed as web application `portal` on the `/portal` URI in the web container.

Overview of APIs

This section describes the Portal Server Desktop, Search, and authentication APIs for extending your portal.

Desktop APIs

The Desktop APIs allow you to create new providers for delivering portal content to users. Conceptually, the Desktop APIs consist of Java interfaces in a "stack" as shown in the following figure:.

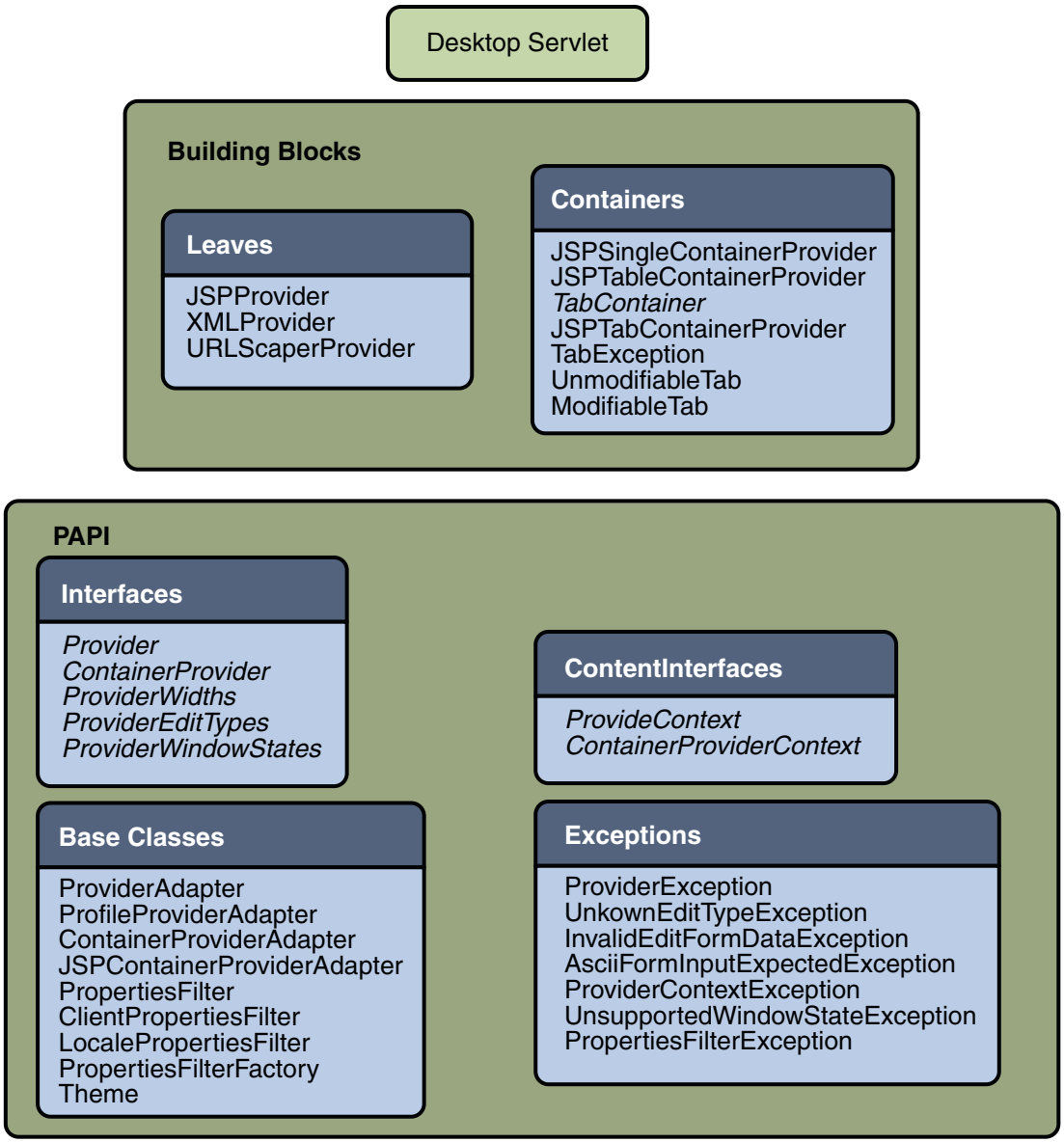

FIGURE 1–3 Desktop APIs

Provider API

At the bottom of Desktop APIs, is the Provider Application Programming Interface (PAPI), a foundation that contains the interfaces, base classes, provider context, and exception classes.

As a developer, use the PAPI and extend the base classes to create new providers. For more information, see Chapters 2 to 4.

The PAPI defines the interface for implementing the provider. A provider is the programmatic entity responsible for generating channels on the Desktop at runtime. The channel properties are read from the display profile by the provider code to dynamically generate the channel content.

There is not necessarily a one-to-one mapping between providers and channels; a single provider can generate one or more channels depending on how you configure it.

Portlet API

The Portlet API version 1.0 is based on the Java 2.0 Platform Enterprise Edition (J2EE) version 1.3. Portlet containers and Portlets meet the requirements, described in the J2EE Specification, for executing in a J2EE environment. The `Portlet` interface is the main abstraction of the Portlet API.

All Portlets implement this interface either directly or, more commonly, by extending a class that implements the interface. The Portlet API includes a `GenericPortlet` class that implements the `Portlet` interface and provides default functionality. If you develop Portlets, you should extend directly or indirectly, the `GenericPortlet` class to implement their Portlets.

The Portlet API defines the `PortletURL` interface. Portlets must create Portlet URLs using `PortletURL` objects. A Portlet creates `PortletURL` objects invoking the `createActionURL` and the `createRenderURL` methods of the `RenderResponse` interface. The `createActionURL` method creates action URLs. The `createRenderURL` method creates render URLs.

Building Blocks

At the next level of "Desktop APIs" on page 31, are the building block providers. Building block providers are those providers that are public and that you can extend to create new providers. The other providers, such as bookmark and mailcheck, are not public and are not extensible.

The building block providers in the figure are all the specific content providers (leaves) and specific container (presentation) providers that Portal Server supplies. All these public building block classes are based upon the base PAPI classes.

As a developer, you can extend the Java classes for some of the building block providers. An administrator can then use your extended classes to define channels for end user consumption.

Desktop Servlet

At the top of "Desktop APIs" on page 31, is the Desktop servlet, which routes client requests for content and processing and passes them on to the specific provider object. The Desktop servlet processes the following actions:

`content`	Gets the named channel's main content
`edit`	Gets the named channel's edit content
`process`	Allows the named channel to process form data
`logout`	Ends the user's session

The action is performed on the channel (for the content, edit, and process actions). The following request parameter names are reserved by the portal Desktop.

- `action`
- `provider`
- `last`
- `containerName`
- `targetprovider`
- `page`
- `error`
- `container`
- `selected`
- `editChannelName`

Note – You cannot extend the Desktop Servlet.

How Concepts in the Provider API Map to the Access Manager Software

The Provider API furnishes architectural separation from the Access Manager; but the PAPI is implemented in terms of specific Access Manager APIs within the Portal Server framework.

Typically, to create a provider, you will need to access various software services for provider development. For example, this might include attribute (property) access, session services, and client-based property retrieval. In the PAPI, these services are accessible through the `ProviderContext` and `ContainerProviderContext` interfaces. The implementation of these interfaces connects to Access Manager services in an implementation independent manner. The actual implementation of software services, for the most part, is located in another layer; the context interfaces simply pull these features together into a common interface to simplify provider development.

Search APIs

The Portal Server Search service provides:

- C API for customizing the way the robot crawls URLs and generates resource descriptions.
- Java APIs for searching the database, for submitting data, and for manipulating Search objects, such as RDs (RDM and Search APIs). C versions of these APIs are also available.
- Search provider taglib and helper beans to write customized search JSPs.

Search Robot

The robot examines a set of selected URLs and searches for documents. For each found document, the robot then creates a resource description (RD) of the document using a predefined schema. The schema defines what pieces of information about the document are put in the RD. For example, the RD could contain a date, the author, the title, the URL, and an abstract about the document. These RDs can be grouped together or classified according to a given hierarchical taxonomy.

You can configure the robot through the Portal Server administration console.

The robot has many customizeable parameters, including the following configuration parameters:

- The URLs that it starts crawling from
- Server access delays
- Passwords
- User agent string
- Certificates for SSL
- Proxy setup

In addition, the robot API enables you to write custom content parsers and summarizers for special URL handling requirements. You can also use the robot API to remove advertisements, generate alerts when certain pages are found, and perform specialized logging.

Search Database

The Search database consists of Summary Object Interchange Format (Search) objects. The search API creates, reads, modifies, and writes the Search database entries. Assisting APIs create buffers, set and get attribute value pairs (used to define content and metadata for the objects in the database), handle exceptions, create a Search output stream, and read a Search input stream.

Normally, the Search database can be accessed by using the Search API, but the database can also be accessed through command-line utilities. You can also add RDs that you create, or import RDs from another database.

An RD is a description of some object to include into the system. Search is the format used to represent RDs.

Authentication APIs

The Portal Server uses the Access Manager APIs for authentication, single sign-on, session, profile, and logging.

In general, most development work for a portal developer in the Access Manager area will be to customize the authentication interfaces. The following table explains the authentication development tasks and where to go for the information.

TABLE 1–1 Authentication Development Tasks

If you want to	Go to
Change the look and feel of the authentication screen	*Sun Java System Access Manager 7 2005Q4 Developer's Guide*
Enable or disable authentication modules	*Sun Java System Access Manager 7 2005Q4 Developer's Guide*
Add a custom authentication module	*Sun Java System Access Manager 7 2005Q4 Developer's Guide*. By default, the Access Manager supplies authentication modules for the following types of logins: Anonymous, Certificate, LDAP, Membership, RADIUS, SafeWord, SecureID, and UNIX.

Application Development

As a developer, you can provide access to portal resources through the Portal Server (and the Access Manager) APIs. For example, you can develop channel content to define aggregation of both those channels as well as channels built from the predefined set into your site's portal.

In extending the Portal Server, use the APIs in the following functional areas:

- Desktop - Using the Provider API, you can create new providers.
- Search service - Search APIs enable you to customize the search robot behavior and manipulate the Search database itself.
- Sun Java System Access Manager - There are five APIs packaged with the Access Manager: authentication, debugging, logging, client detection, and Single Sign-on (SSO). You can extend the authentication and client detection APIs. The other APIs that can be used include the functional areas of SSO, debugging, and logging. This guide does not describe the Access

Manager APIs in detail, but instead provides a general overview. See theAccess Manager documentation for more information on programming with those APIs.

Display Profile

The display profile is a set of XML documents used to define and configure providers and channels in the Portal Server. The display profile defines:

- Providers
- Channels
- Container channels

A provider's display profile document acts as a template for creating channels. They define the set of properties that channels based on this provider will make use of, as well as providing default values for these properties where appropriate. Channels and container channels must reference a provider, and will use their default property values unless the property is redefined in the channel.

The display profile used to generate a user's Desktop is constructed by merging together multiple display profile documents. Each display profile contains a series of XML instructions for storing channel properties.

The display profile documents are stored in their entirety as a single attribute in the Sun Java System Access Manager services layer. That is, the display profile documents are an LDAP attribute residing in an instance of the Sun Java System Directory Server.

See the *Sun Java System Portal Server7.1 Technical Reference Guide* for a complete discussion of the display profile.

◆ ◆ ◆ CHAPTER 2

Overview of the Provider API (PAPI)

This chapter provides an overview of the Sun Java System Portal Server software Provider Application Programming Interface (PAPI).

This chapter contains the following sections:

Introduction to the Provider API

The following figure shows the relationship between the various interfaces, classes, and exceptions discussed in this chapter. For detailed information on these interfaces, classes, and exceptions, see the Javadocs at `/opt/SUNWportal/sdk/desktop/javadocs.jar`.

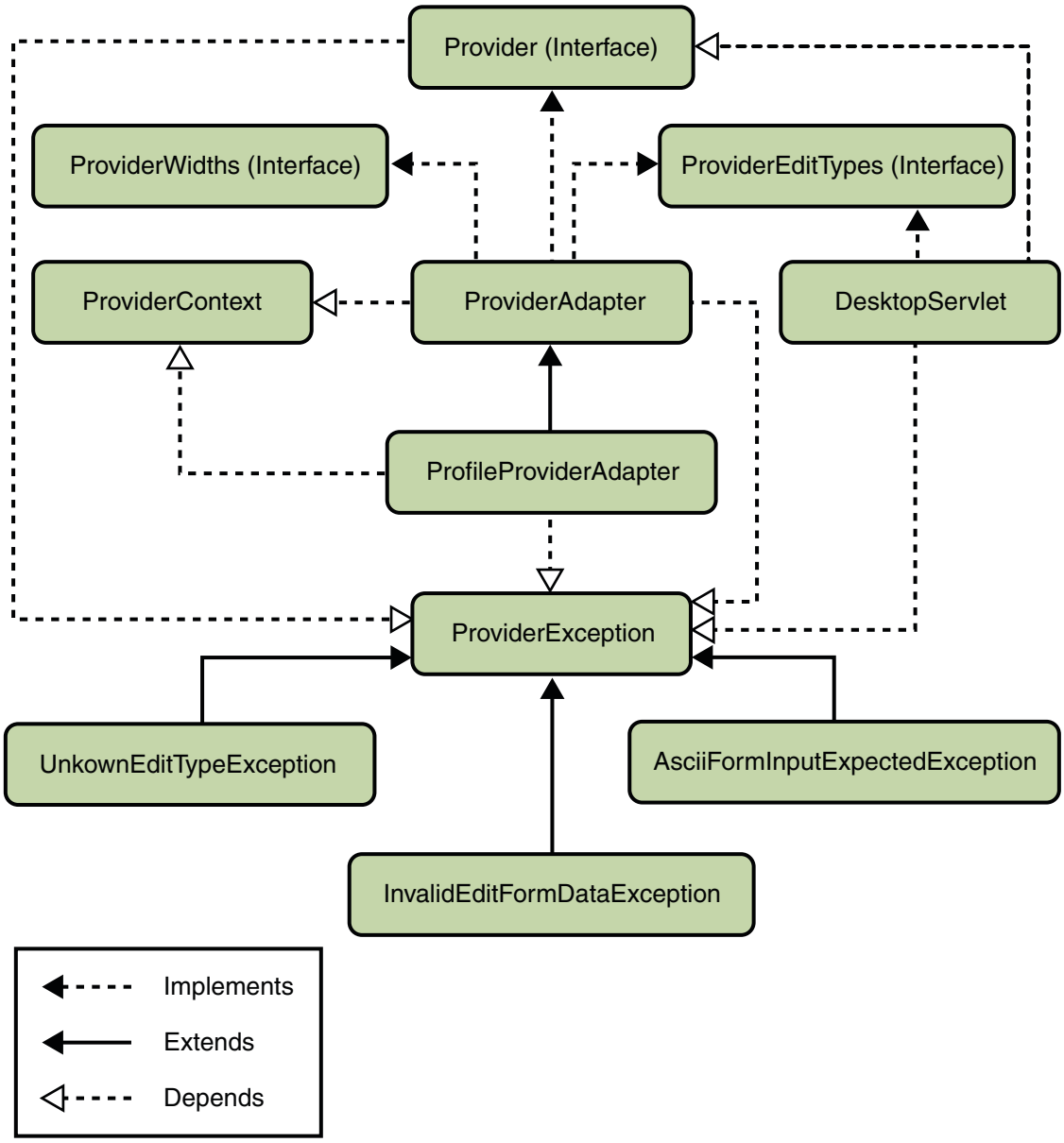

FIGURE 2–1 The Provider API

The Provider API

This section provides an overview of the interfaces (see Provider, ProviderWidths, and ProviderEditTypes), base classes (see ProviderAdapter, ProfileProviderAdapter), context (see ProviderContext), exceptions (see Exceptions), and the lifecycle of the provider.

Provider Interface

The Provider interface clearly defines the communication channel between a container and a provider. The container may be the DesktopServlet, or another provider object. The methods defined in the Provider interface supply the basic handshaking required for a container to display the content from a provider object.

A provider can implement this interface. Clients of the Provider interface call the methods in this interface to query information or to perform an action in the provider. Such clients include, but are not limited to, the DesktopServlet and other channels (container channels).

There are essentially two types of methods in this interface:

Methods that handle actions
: This includes `getContent()`, `getEdit()`, and `processEdit()` methods. These methods control any channel content generation and are called by the DesktopServlet or the channel's parent container.

Methods that describe how the provider should be treated by the containing entity
: This includes all methods that begin with `is` and `get` (minus the `get*` methods used for handling actions).

See the Javadocs for more information on the methods in this interface.

ProviderAdapter Class

The ProviderAdapter class implements aspects of the Provider interface using the ProviderContext. This class implements the accessor and mutator methods of the Provider interface that access the mandatory provider properties. It uses the ProviderContext interface (which contains functionality to read and write properties in the display profile) as its persistent store.

The ProviderAdapter class implements the ProviderWidths interface, which defines the provider widths, and ProviderEditTypes interface, which defines the edit types, making the constants defined in these interfaces available to classes that extend ProviderAdapter.

Developers who wish to implement a provider can extend this class or the ProfileProviderAdapter class for forward compatibility and convenient access to the ProviderContext interface.

See the Javadocs for more information on the methods in this class.

ProfileProviderAdapter Class

The ProfileProviderAdapter is a subclass of the ProviderAdapter that includes convenience wrappers around some commonly used methods in the ProviderContext interface, like `get/set*Property()`, `exists*Property()`, `get/setStringAttribute()`, and `getTemplate()`.

The advantages of using the wrapper methods in the ProfileProviderAdapter class as opposed to using the methods in the ProviderContext interface are the following:

- For example, the `ProviderContext.getStringProperty()` accepts the channel name as an argument. Since the ProfileProviderAdapter knows the channel name, it can use `ProfileProviderAdapter.getStringProperty()` method without providing the channel name.
- All of the methods in ProviderContext throw a ProviderContextException. The analogous methods in ProfileProviderAdapter throw a ProviderException. For example, when implementing the `getContent()` method, since this method is already defined to throw a ProviderException, when a client calls `ProfileProviderAdapter.getStringProperty()` inside the `getContent()` method, you need not implement a `try-catch` block. However, if you use `ProviderContext.getStringProperty()`, you must catch the ProviderContextException.

See the Javadocs for more information on the methods in this class.

ProviderContext Interface

The ProviderContext interface is the runtime environment that providers use to retrieve essential information from the Portal Server. The ProviderContext object provides information pertaining to the environment that a provider object is executing within. Such information may be specific to the user, to the web container, or be global (shared between multiple users). This interface does not define what information falls into each of these categories; this is left up to the implementation. Provider developers can obtain a handle to a ProviderContext object by extending ProviderAdapter, and then calling `ProviderAdapter.getProviderContext()`.

The ProviderContext forms the layer between the services used by the provider and the provider implementation thus isolating the provider code from specific service interfaces and implementations. It defines the service interface for the Desktop, and allows different implementation to access the actual data.

The ProviderContext methods are:

- Service (data/functions pertaining to the Desktop service): `getDesktopURL()`, `getLocaleString()`, `getLocale()`, `getDesktopType()`, `getLogoutURL()`, `getStringAttribute()`, `setStringAttribute()`, `getDefaultChannelName()`, `getLoginURL()`, `getRoles()`
- Servlet (data/functions pertaining to the servlet): `getRequestServer()`, `getServletConfig()`
- Client type (data/functions pertaining to the client device type): `getClientTypeProperty()`, `getDefaultClientType()`, `getClientType()`, `getCharset()`, `getClientPath()`, `getContentType()`, `getClientTypeProperties()`
- Session:
  - Data/functions pertaining to the user's session: `get/setSessionProperty()`, `getSessionID()`, `getUserID()`, `encodeURL()`, `isAuthless()`
  - Data/functions pertaining to per-client properties: `get/setClientProperty()`
  - Data/functions pertaining to the debug service: `isDebugEnabled()`, `isDebugErrorEnabled()`, `isDebugWarningEnabled()`, `isDebugMessageEnabled()`, `debugMessage()`, `debugWarning()`, `debugError()`
  - Data/functions pertaining to template access: `getTemplate()`, `getTemplatePath()`, `getTemplateMostSpecificPath()`
  - Data/functions pertaining to channel properties: `getProviderName()`, `getClassName()`, `getNames()`, `get/set*Property()`, `exists*Property()`, `getProviderVersion()`
  - Data/functions pertaining to configuration properties: `getStaticContentPath()`, `getConfigProperty()`

  Data/Functions pertaining to ClientTypeFilters: `getClientPropertiesFilters()`, `getLocalePropertiesFilters()`, `getClientAndLocalePropertiesFilters()`
- Data/Functions pertaining to URLEncoding: `encodeURLParameter()`

See the Javadocs for more information on the methods in this interface.

ProviderWidths Interface

The ProviderWidths interface defines the widths that can be returned from the `Provider.getWidth()` method. The width is a suggestion to a client of a provider object as to how much screen real estate should be given to display the provider's default view.

- The provider can be displayed in a thick frame (`WIDTH_THICK`).
- The provider can be displayed in a thin frame (`WIDTH_THIN`).
- The provider can be displayed in a full_top frame at the top (`WIDTH_FULL_TOP`).
- The provider can be displayed in a full_bottom frame at the bottom (`WIDTH_FULL_BOTTOM`).

See the Javadocs for more information on the methods in this interface.

ProviderEditTypes Interface

The ProviderEditTypes interface defines the edit types that can be returned from the *Provider*.`getEditType()` method. The edit type informs a client of a provider object what it can expect to be returned from the provider's `getEdit()` method. The edit type can be `EDIT_COMPLETE` or `EDIT_SUBSET`.

- `EDIT_SUBSET` - Indicates that the edit page is not a complete document. This value is potentially returned from `getEditType()` to signify that the buffer returned is only a subset of a document. The edit type, if subset, means that the JSP edit container is used to wrap the content from the container. This is useful for providing a common look and feel for a set of common portal pages. That is, it allows the channel's edit page to look and feel according to the container that it exists within although the same channel might appear in two different containers that have very different look and feel. In this case the form that wraps the edit content generated by the provider is drawn by the edit container.
- `EDIT_COMPLETE` - Indicates that the edit page is a complete document. This value is potentially returned from *Provider*.`getEditType()` to signify that the buffer returned is a complete document. The container of the provider uses this information to determine if it needs to wrap it's edit page output to form a complete document. This is done with something called an edit container.

  Containers use edit containers to provide a container-specific look and feel to otherwise heterogeneous providers. It allows the channel's edit page to look and feel according to the container that it exists within although the same channel might appear in two different containers that have very different look and feel.

See the Javadocs for more information on the methods in this interface.

PropertiesFilter Class

The PropertiesFilter abstract class, when extended, can describe a specific filter criteria. To implement a specific filter, one must minimally implement the `getCondition()` and `match()` methods in the PropertiesFilter abstract class.

- The `getCondition()` method must return the condition on which the PropertiesFilter should operate.
- The `getValue()` method must return the value that corresponds to the given condition.
- The `isRequired()` method can be implemented to specify whether you want this to be a required filter or not. A conditional property lookup involves one or more property filters. If a filter in the filter list is required, then it must match for the overall conditional lookup to succeed. If a filter is not required, then it can fail to match without causing the overall lookup to fail.

A chain of non-required filters can be used to implement a progressively less-specific filter lookup, similar to the semantics of Java resource bundle lookup. For instance, an optional filter would be useful in a case where a locale lookup is followed by a date lookup. Given the filter `{locale=en, locale=US, date=03/03/2003}`, you can get it to successfully match a property with the qualifier `{locale=en; date=03/03/2003}` even though it does not exactly match the filter specification. This can be accomplished by setting the locale filter to be optional. The `locale` property type is deprecated.

- The `match()` method returns true if this filter object agrees with or matches the condition and value that are passed in, given the information that was used to create the object.

Out of the box, the Portal Server software includes filters based on locale and client. The locale and client filter extend the PropertiesFilter abstract class and includes an implementation of the `getCondition()` and `match()` methods in the PropertiesFilter class. See the Javadoc for more details.

TypedException

TypedException is a Public Application Programming Interface (API), which the users can implement to create a user defined exception class. This interface helps to generate context related error messages rather than generic error messages. The user exception that implements TypedException can be thrown from the JSP incase of any error and the desktopservlet can display the proper error template according to the exception type set in the user exception.

The exception types defined in the TypedException interface are:

- public static final String *NON_EDITABLE_TYPE* = "nonEditableChannel";
- public static final String *UNKNOWN_CHANNEL_TYPE* = "unknownChannel";
- public static final String *NO_PRIVILEGE_TYPE* = "noPrivilege";
- public static final String *SESSION_TYPE* = "session";
- public static final String *UNKNOWN_TYPE* = "unknown";
- public static final String *SESSION_TIMED_OUT* = "sessionTimedOut";

Provider Exceptions

The Desktop expects a provider to only throw ProviderException or a subclass of the ProviderException. For correct operation, a provider must only throw expected exception type. That is:

- Minor exceptions can be managed internally, for example, log a debug message.
- Serious exceptions can be rethrown as an expected exception type.

Exceptions from providers are logged in *PortalServer-DataDir*`/portals/`*portal-ID*`/logs/`*portal-instance*`/portal.0.0.log` file. Note that this file is created only if there is an error.

ProviderException Class

The ProviderException is a generic superclass for all provider related exceptions.

AsciiFormInputExpectedException Exception

The AsciiFormInputExpectedException will be thrown from *Provider*`.processEdit()` method when something other than ASCII only encoded form input is sent to it.

InvalidEditFormDataException Exception

The InvalidEditFormDataException is thrown from the *Provider*`.processEdit()` method when there is an error in the data input by the user. If thrown, the Desktop will send back the same Edit page, and will attach the exception's message as a parameter to the URL. For example, if the exception is:

```
throw new InvalidEditFormDataException("Error Error");
```

the Desktop will redirect back to the same Edit page, adding the error message to the URL error parameter:

```
error=Error Error
```

The edit page wrapper then looks for the error parameter in the URL and if present, displays the message (or the value of the error parameter) at the top of the page in red.

UnknownEditTypeException Exception

The UnknownEditTypeException may be thrown from *Provider*`.getEditType()` method if an unknown or undefined edit type is encountered.

The Provider Life Cycle

This section describes the lifecycle of the provider per user session. Figure 2–2, Figure 2–3, and Figure 2–4 shows how the Desktop handles each user session.

A session is created at the time the user logs in to the portal Desktop and ends when they logout. The session also ends when the user session is idle, or when the session times out. For authless logins, a single session is shared amongst all clients accessing the Desktop in authless mode, and the authless session never dies.

For each request, the DesktopServlet will validate the user session; if it is a valid session, the DesktopServlet will start to process the request. If a session does not exist, the DesktopServlet will create a session.

The creation of the provider context is equivalent to the creation of a session. The container provider context object is generated and maintained, one per session. So each time a request comes in, the cached container provider context object is used. For example, in Figure 2–2, when the first initial request comes in, the container provider context is created and initialized, and when the subsequent requests come in, as in the reload and logout requests shown in Figure 2–3 and Figure 2–4 respectively, the cached container provider context is returned.

The container provider context maintains a list of its contained provider objects and their cached content locally. Contained providers are created and initialized once per session, and their content is cached in the provider context object. Subsequent requests access the cached provider objects that live inside the provider context object.

The `contentChanged()` and `allContentChanged()` methods signify that either the content Chapter 3 for one channel or all channels has changed, respectively. This is used to remove all the cached content for the channels, as well as to clear the cached provider objects. When the requested action is logout, as shown in Figure 2–4, the DesktopServlet will redirect the request to the logout URL, and the session will be destroyed. At this point, the provider context object is removed from the cached list, and ready for garbage collection. Also, if the session expires or times out, the provider context object is removed from the cached list.

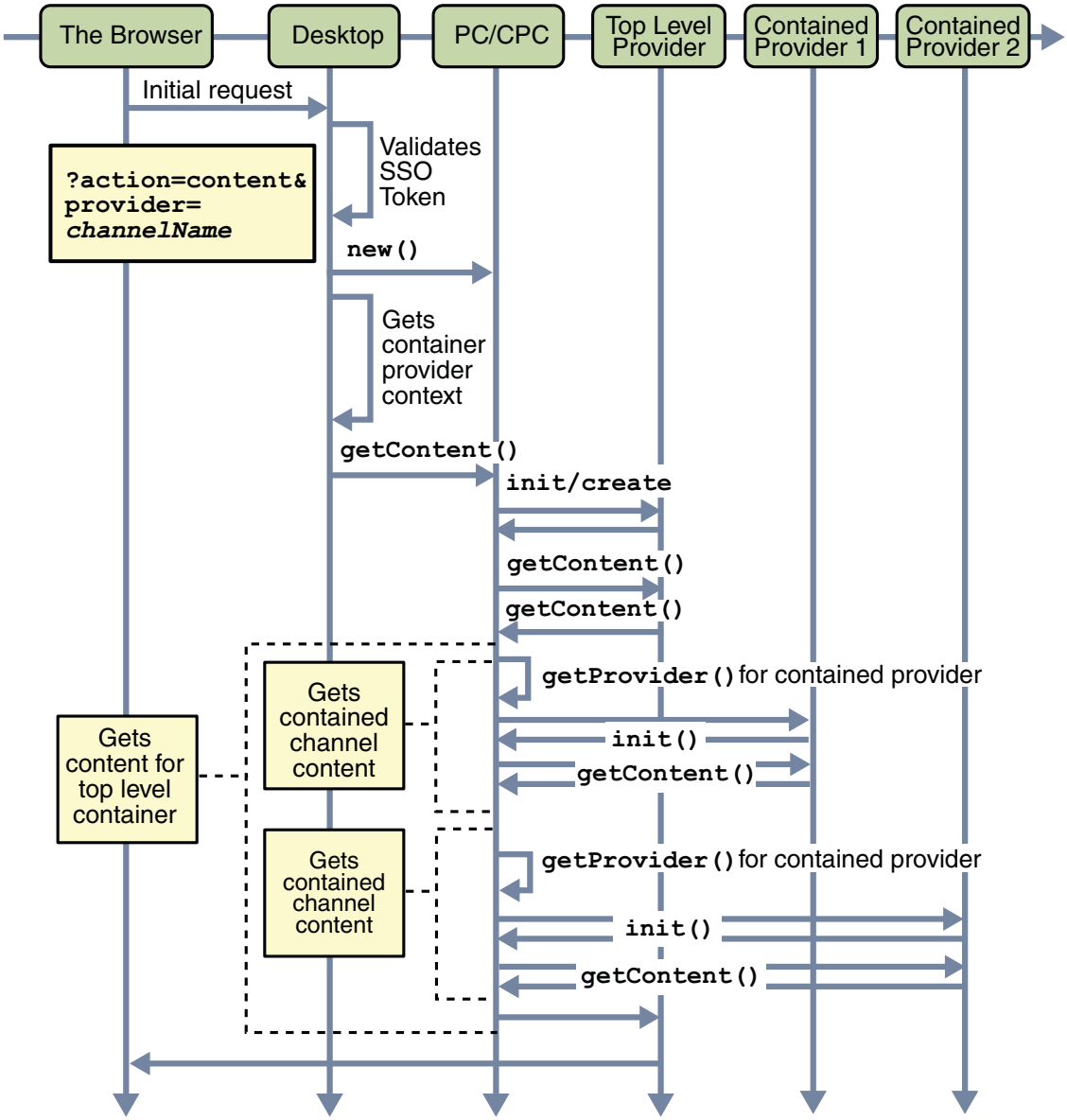

FIGURE 2–2 The Provider Life Cycle - Initial Request for Authenticated User

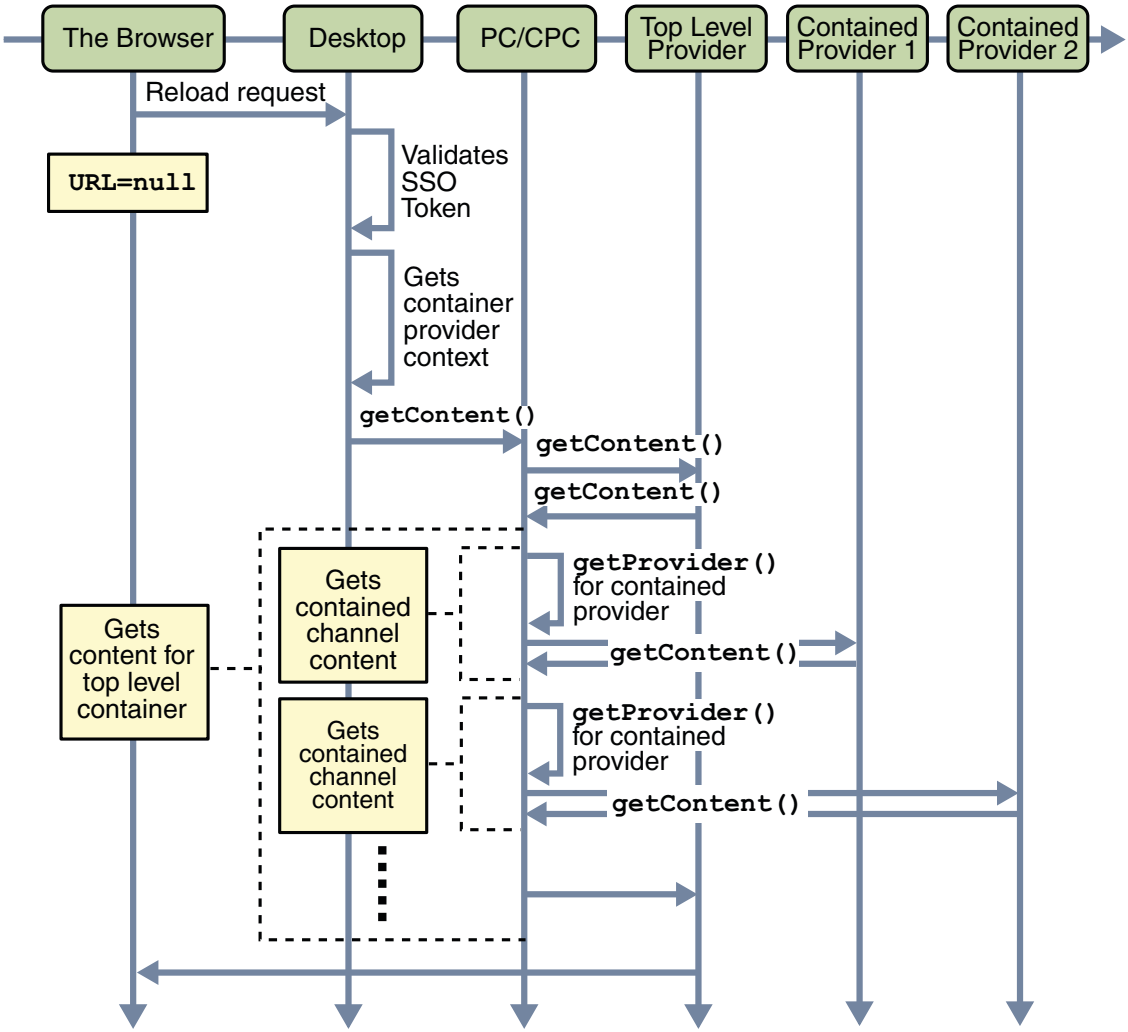

FIGURE 2–3 The Provider Lifecycle - Desktop Reload Request for Authenticated User

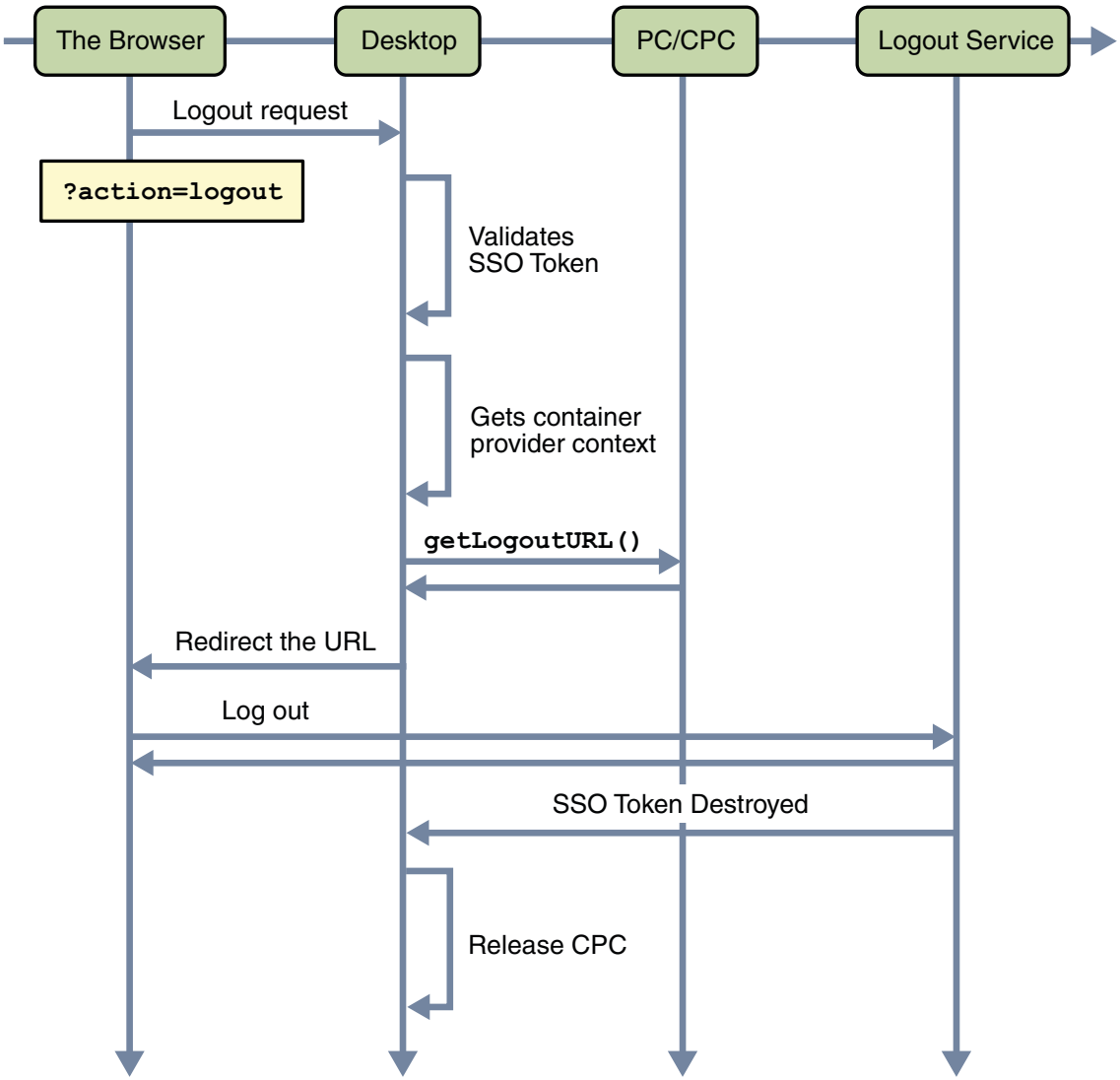

FIGURE 2–4 The Provider Lifecycle - Logout Request

Figure 2–2 and Figure 2–4 shows a sample of how the content and logout actions are processed. For detailed information on how the DesktopServlet and the back end providers handle each action (including content and logout), see Chapter 3.

◆ ◆ ◆ CHAPTER 3

Overview of the Desktop Servlet

This chapter provides an overview of the DesktopServlet and discusses the relationship between the DesktopServlet and the PAPI. It provides information on how the DesktopServlet uses the various methods in the PAPI to perform the various actions (such as `content`, `edit`, `process`, and `logout`).

This chapter contains the following sections:

Introduction to DesktopServlet

The DesktopServlet coordinates the drawing of the Desktop, dispatches the process to the target channel based on the information stored in the underlining services, and validates the user with the Sun Java System Access Manager software.

In this sense, the DesktopServlet is a router of requests. It catches requests for content and processing, and passes them on to the specific provider object. Whenever a provider throws an exception that cannot be handled by the container provider, the exception will propagate all the way up to the DesktopServlet, and the DesktopServlet will display an error page.

For detailed information on how the DesktopServlet creates and validates a user session and creates and gets the provider context object, see the "The Provider Life Cycle" on page 46. The following sections describe how the DesktopServlet handles various actions.

DesktopServlet Actions

The DesktopServlet understands several actions. Every action has an associated channel or container name and actions are performed on the associated channel or container. Actions are passed to the servlet via request parameters. The associated channel name is also passed as a parameter.

For example, to perform an action on a channel, provide the following parameters to the servlet:

DesktopServletURL`?action=`*actionType*`&provider=`*ChannelName*

Here:

- `action` - indicates the type of action to take. Action can be content, edit, process, or logout.
- `provider` - indicates the name of the provider to contact. The provider argument is named that way for historical reasons; the value of the provider argument is really a channel or container name. If the provider parameter is absent or null in the request, the servlet assumes it is equal to the value of the last request. For the initial request, if the provider parameter is absent, then the `defaultChannelName` from the Desktop service will be used.

The action and provider parameters are not required; if they are absent, the default action is content, and provider is the value set in the Desktop service for the default channel.

The `content`, `edit`, and `process` actions map directly to method calls into the PAPI. For Desktop actions that map to PAPI method calls, the servlet passes an HTTP request and response object to the respective provider method. These objects are not the same objects passed into the servlet. The request and response objects passed to provider objects from the servlet are wrappers around the original request and response passed into the servlet. This is because there is certain functionality that is available from request and response objects that is not applicable to a provider. See the Javadocs for the Provider interface for more information.

The HTTP parameters in the original request object are processed before they are copied to the wrapper servlet request and response objects. As part of this processing, the parameters are decoded from the character set encoding used to represent the page into Unicode that is used in Java String objects. Therefore, the parameters that are passed to the providers are all stored as Unicode, and the provider does not have to do any decoding of its own related to the character encoding for the page.

Action content

When the action is `content`, the DesktopServlet gets the named channel's main content. When the DesktopServlet receives a request where the action is `content`, to perform the `content` action on the channel, it takes the following parameters:

DesktopServletURL`?action=content&provider=`*ChannelName*`[&last=false]`

The `content` action maps directly to the following method calls in the PAPI: `ProviderContext.getDefaultChannelName()`, *Provider*`.isPresentable()`, and *Provider*`.getContent()`.

The flowchart in Figure 2-5 on page 39 shows the various methods executed in the back end to process the content action. When the client makes a request for content (say after login), the DesktopServlet:

1. Determines the provider responsible for generating the requested content.

   If provider is null, it uses the default channel name (stored in the `DefaultChannelName` attribute) to get the provider. The default channel name is an Sun Java System Access Manager software attribute (`DefaultChannelName`) in the Desktop service, which is set to the topmost container that represents the whole Desktop view. The default channel name is set to the current target provider value when the request parameter last is set to `false`. If the provider is not null, the DesktopServlet gets the provider responsible from the HTTP parameter in the URL for generating the requested content.

2. Once the provider is determined, the provider's `isPresentable()` method is invoked to determine if the provider can be presented to the requesting client.

   If the provider is determined to be not presentable, an error is thrown on the Desktop. If the provider can be presented, the provider's `getContent()` method fetches the content for display on the client's Desktop.

3. When last is set to `false`, the DesktopServlet will not set the last channel to the value of the provider parameter. If not specified, the default setting is `last=true`, and the last accessed channel is set to the value of the provider parameter.

   The next time when the action is `content`, the DesktopServlet provider parameter will be used to fetch the content. If the `provider` parameter is absent in the request URL, the DesktopServlet assumes it is equal to the value of the last request.

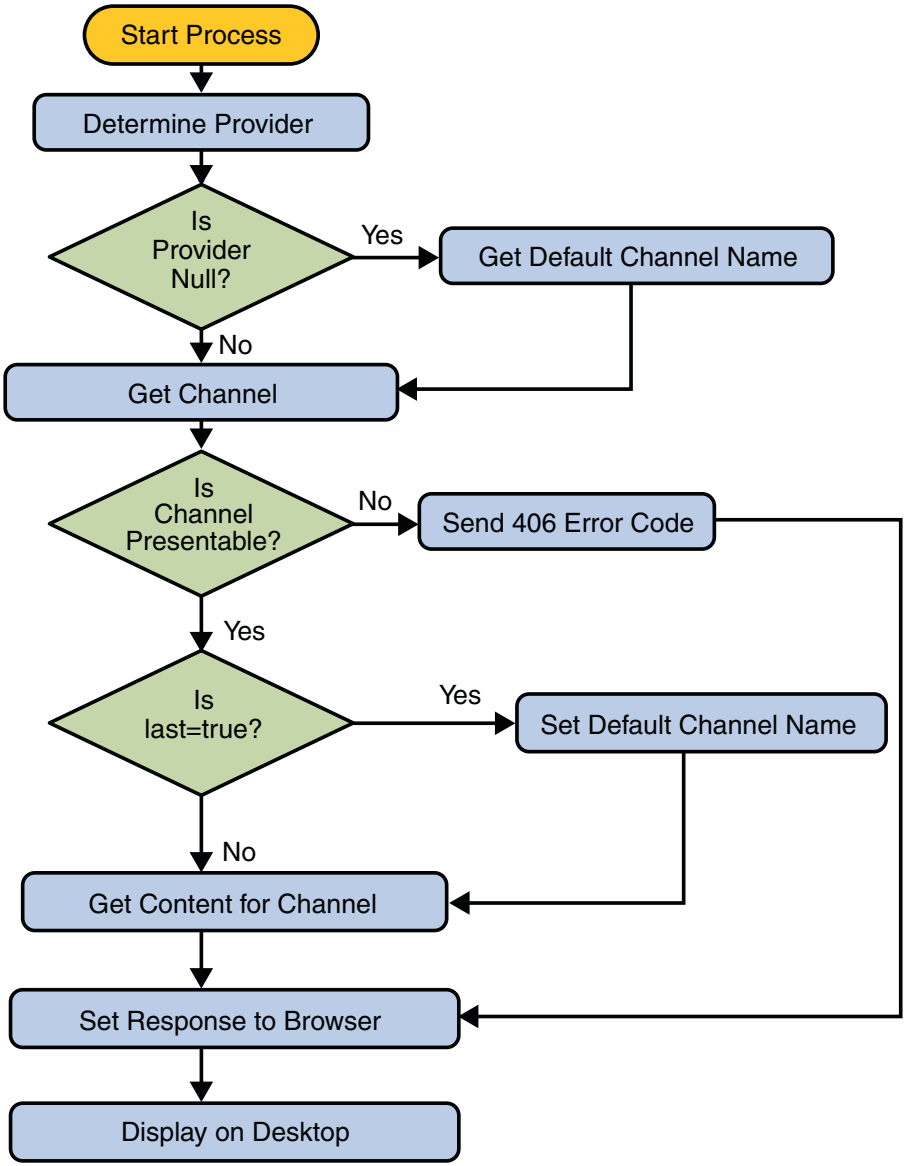

FIGURE 3–1 DesktopServlet content Action

Action edit

When the action is `edit`, the DesktopServlet gets the named channel's or edit container's request parameters and starts to process the `edit` action. When the DesktopServlet receives a request where the action is `edit`, to perform the `edit` action, it takes the following parameters:

DesktopServletURL`?action=edit&provider=`*ChannelName* for backward compatibility

Or,

DesktopServletURL`?action=edit&provider=`*editContainer*`&targetprovider=`*ChannelName*

The DesktopServlet checks the edit types based on the values defined in the ProviderEditTypes interface.

DesktopServlet Legacy edit Action

DesktopServletURL`?action=edit&provider=`*ChannelName*

The flowchart in "Action edit" on page 54 shows the various methods executed in the back end to process the edit action. When the client makes a request to edit the channel's editable parameters, the DesktopServlet:

1. Determines the provider responsible for generating the requested content.

   If provider is null, it uses the default channel name to get the provider. Last accessed channel is not assumed when action equals `edit`. Otherwise, the DesktopServlet gets the provider responsible specified in the URL.

2. Once the provider is determined, the channel's `isEditable()` method is invoked to determine if the channel is editable.

   The DesktopServlet gets the named channel's `isEditable` property. If the channel is determined to be not editable, an exception is thrown on the Desktop. If the channel can be edited, the DesktopServlet also checks the `editType` for that channel.

   The `editType` is a channel property that can be retrieved via the *Provider*`.getEditType()` method. The provider's `getEditType()` method is invoked to determine the type of edit page to return on the Desktop.

3. If the channel's edit type is:

   - `EDIT_COMPLETE`, the provider's `getEdit()` method is invoked and the Edit page for the channel is returned on the Desktop.
   - `EDIT_SUBSET`, the edit container's provider name will be detected (via the default Desktop Edit Container attribute), and then the edit container's `getEdit()` method will be invoked. Also:

     The edit container's provides a common look and feel of the edit page for all the channels that it contains. After it generates the markup for the common look and feel, it detects the channel name from the request parameter and determines whether the target channel is editable.

     If the target channel is not editable, it throws a provider exception. If the target channel is editable, it gets the channel's edit type and delegates the process to the target channel's `getEdit()` method.

If the channel's edit type is:

- `EDIT_COMPLETE`, the Edit page is displayed.
- `EDIT_SUBSET`, the edit container's content is displayed and it wraps around the channel's edit content.

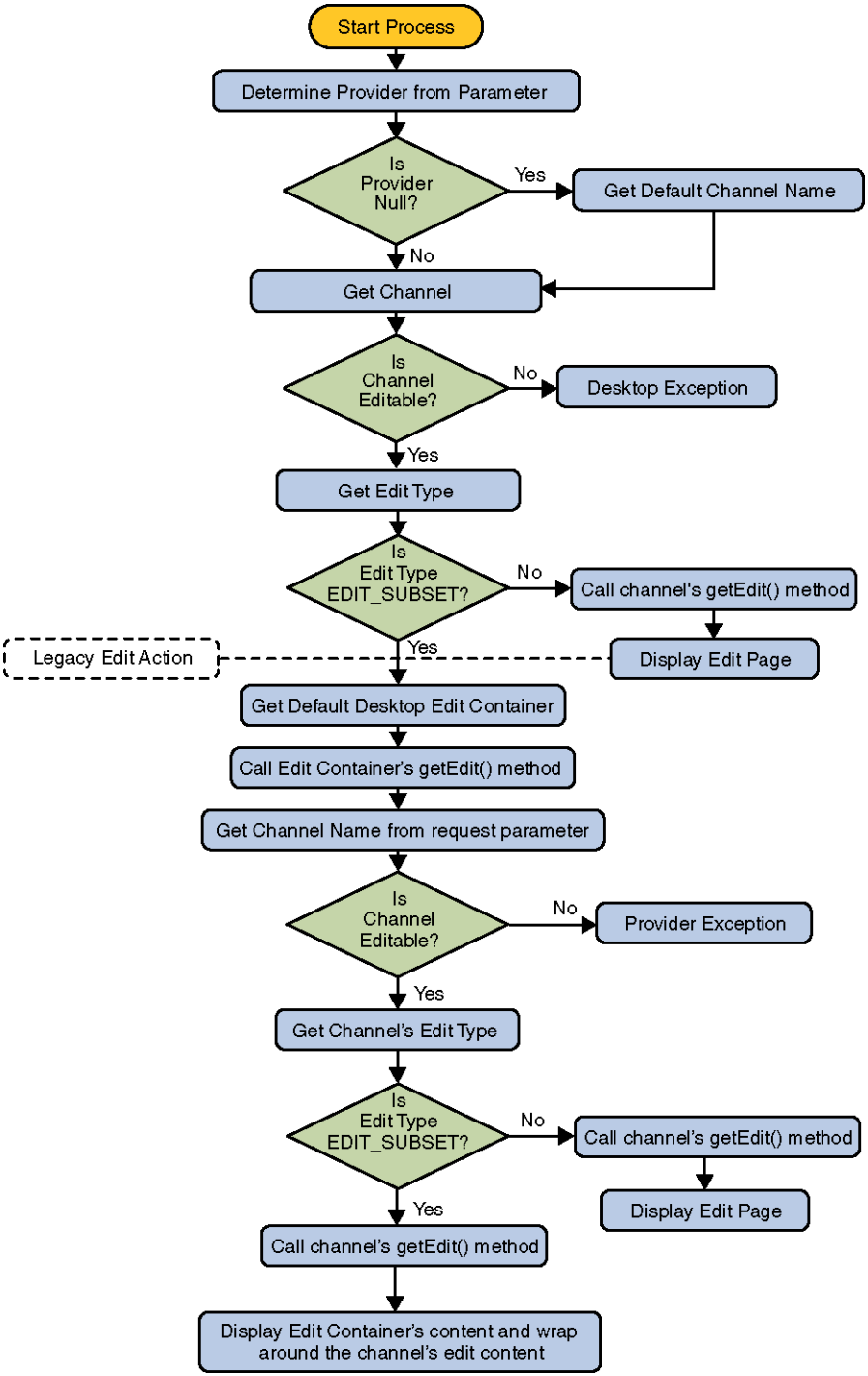

FIGURE 3–2 DesktopServletLegacyEditAction

DesktopServlet edit Action

DesktopServletURL`?action=edit&provider=`*editContainer*`&targetprovider=`*ChannelName*

The flowchart in "Action edit" on page 54 shows the various methods executed in the back end to process the edit action. This flowchart shows how the DesktopServlet processes the edit action for a container. In this URL, the `provider` parameter specifies the edit container for the container, and the `targetprovider` parameter is the leaf channel inside the container. When the client makes a request to process this edit action, the DesktopServlet:

1. Determines the provider responsible for generating the requested content.

   If provider is null, it uses the default channel name to get the provider. Last accessed channel is not assumed when action equals `edit`. Otherwise, the DesktopServlet gets the provider responsible specified in the URL.

2. Once the provider is determined, the edit container's `isEditable()` method is invoked to determine if the edit container is editable.

   The DesktopServlet gets the named edit container's `isEditable` property. If the edit container is determined to be not editable, an exception is thrown on the Desktop. If the edit container can be edited, the DesktopServlet also checks the `editType` for that edit container.

   The `editType` is a property that can be retrieved via the *Provider*`.getEditType()` method. The provider's `getEditType()` method is invoked to determine the type of edit page to return on the Desktop.

3. If the edit container's edit type is:
   - `EDIT_COMPLETE`:
     - The edit container's `getEdit` method is invoked and the leaf channel's name is determined from the request parameter.
     - Once the leaf channel's name is determined, the channel's `isEditable()` method is invoked to determine if the channel is editable.

       If the channel is determined to be not editable, an exception is thrown on the Desktop. If the channel can be edited, the DesktopServlet also checks the `editType` for the leaf channel.
     - If the leaf channel's edit type is:
       - `EDIT_COMPLETE`, the container calls the leaf channel's `getEdit()` method and displays the edit page on the Desktop.
       - `EDIT_SUBSET`, the container calls the leaf channel's `getEdit()` method and the container's Edit Container wraps the leaf channel's content before displaying the Edit page ont the Desktop.

         A container can define different edit container and the edit container name must be specified as well as the target channel name. The default Desktop edit container name is stored in the Access Manager as a Desktop service attribute.

- `EDIT_SUBSET`, the Desktop servlet fetches the name of the edit container to use to wrap the channel's edit page. The servlet fetches a handle to the edit container, and then calls the edit container's `getEdit()` method. The edit container detects the original channel's name from request parameters and calls `getEdit()` on that channel, and wraps the channel's content before returning it to the servlet. See the legacy edit action for `EDIT_SUBSET` in "Action edit" on page 54 for more information.

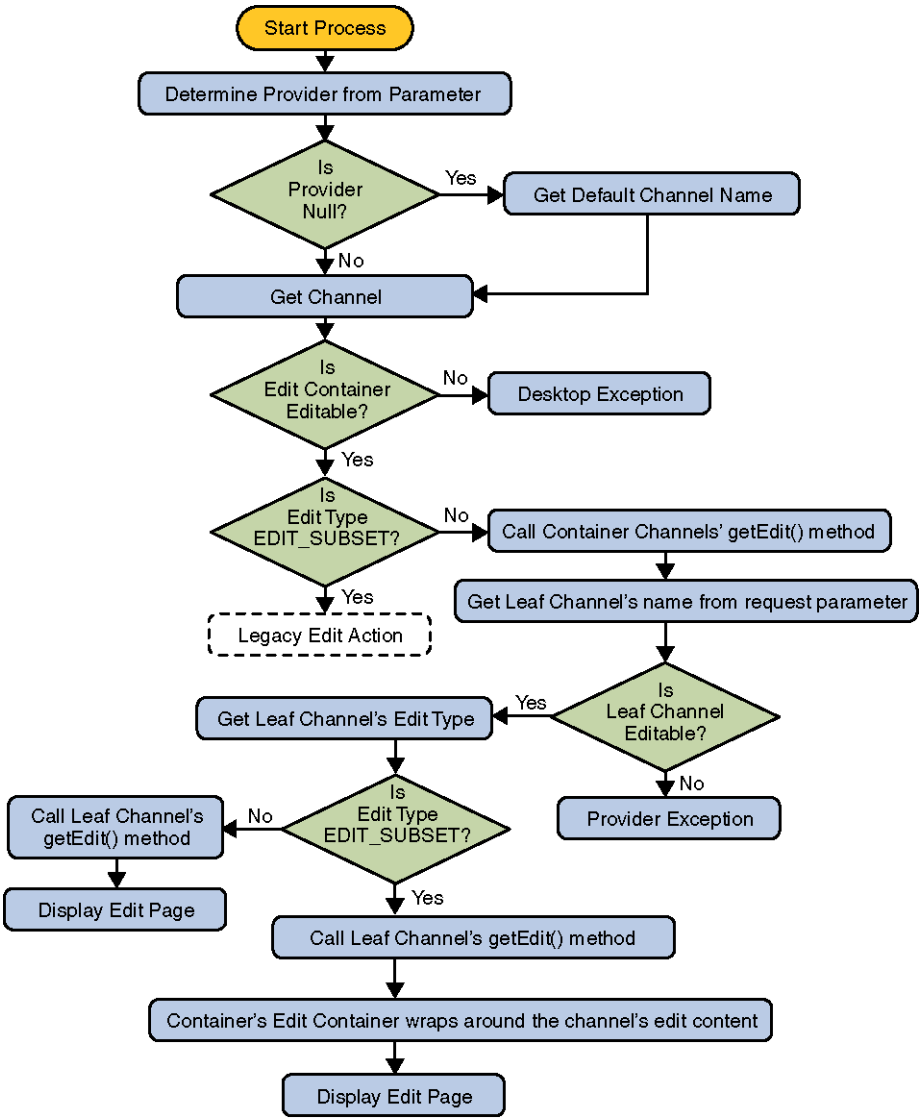

FIGURE 3–3 DesktopServlet edit Action

Action process

The `process` action allows the named channel to process URL parameters and form data, typically that of the channel's edit form. When the DesktopServlet receives a request where the action is `process`, to perform the `process` action, it takes the following URL parameters:

DesktopServletURL`?action=process&provider=`*channelName*

Or,

`DesktopServletURL?action=process&provider=`*editContainer*`&targetprovider=`*channelName*

DesktopServlet Legacy process Action

`DesktopServletURL?action=process&provider=`*channelName*

When the DesktopServlet receives a request where the action is `process` (see "Action process" on page 60), the DesktopServlet:

1. Looks at the parameters to identify which provider will handle the action, through the provider's `processEdit()` method.

   The `processEdit()` method is called to process the edit page generated from the `getEdit()` method. The request passed in contains the parameters.

2. Re-directs to the URL returned from the provider's `processEdit()` method.

   If there is an InvalidEditFormDataException, the DesktopServlet will redirect the browser back to the channel's edit page and include a URL parameter error so that the channel may display the cause of the exception to the user. That is, the DesktopServlet will get the error message and generate a new request as follows.

   DesktopServletURL`?action=edit&provider=`*channelName*`&error=`*errormessage*

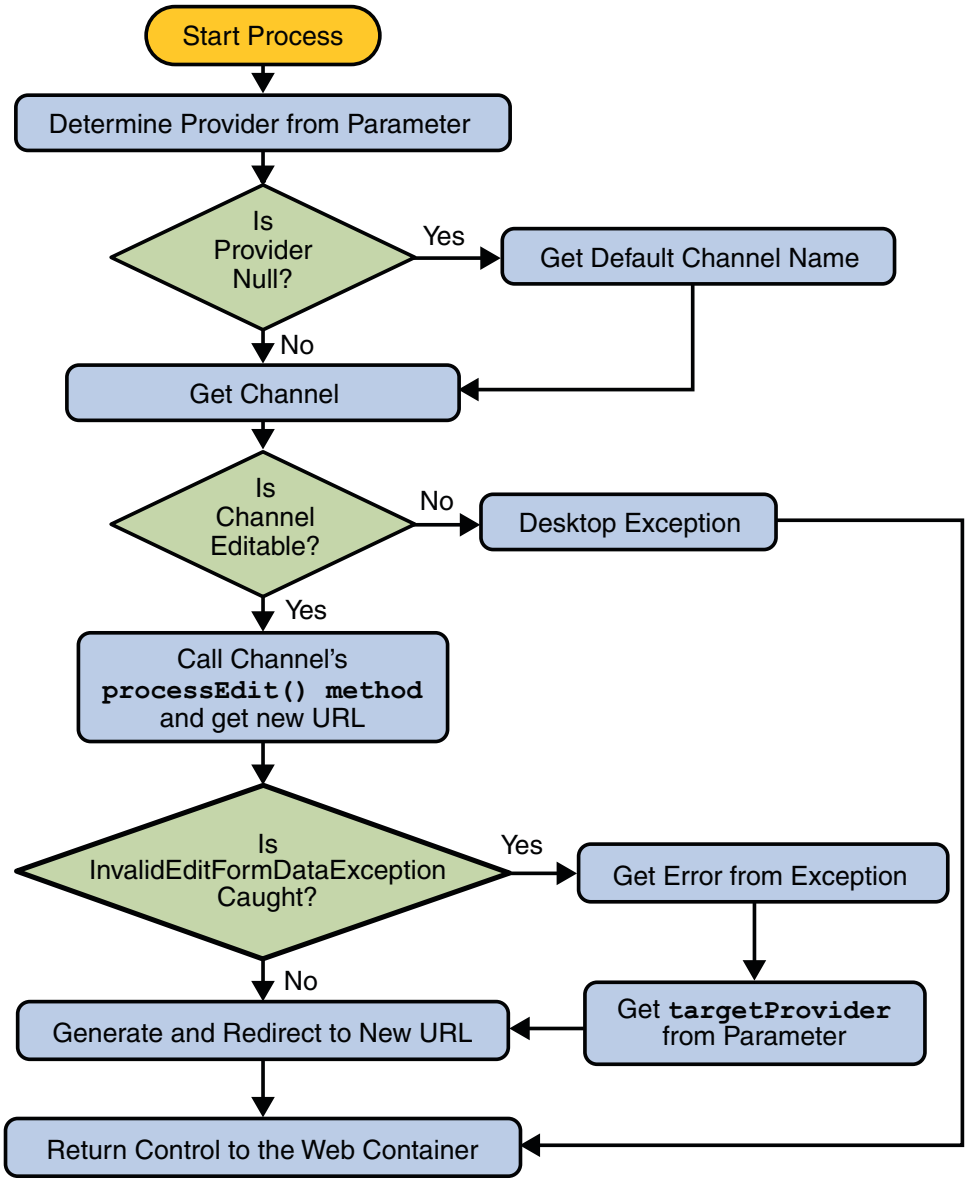

FIGURE 3–4 DesktopServlet Legacy process Action

DesktopServlet process Action

`DesktopServletURL?action=process&provider=`*editContainer*`&targetprovider=`*channelName*

In this URL, the `provider` parameter specifies the edit container for the container, and the `targetprovider` parameter is the leaf channel inside the container. When the DesktopServlet receives a request where the action is `process` (see "Action process" on page 60), the DesktopServlet:

1. Determines whether the edit container is editable.

   If the edit container is not editable, an exception is thrown on the Desktop and control is returned to the web container. If the edit container is editable, it calls the edit container `processEdit()` method and determines the channel's name from the request parameter.

2. Once the channel name is determined, it determines whether the channel is editable.

   If the channel is not editable, it throws a provider exception. If the channel is determined to be editable, it calls the channel's `processEdit()` method.

3. Re-directs to the URL returned from the provider's `processEdit()` method.

   If there is an InvalidEditFormDataException, the DesktopServlet will redirect the browser back to the channel's edit page and include a URL parameter error so that the channel may display the cause of the exception to the user. That is, the DesktopServlet will get the error message and generate a new request as follows:

   DesktopServletURL`?action=edit&provider=`*channelName*`&error=`*errormessage*

   If there is no InvalidEditFormDataException, the DesktopServlet generates the new URL and returns control to the web container.

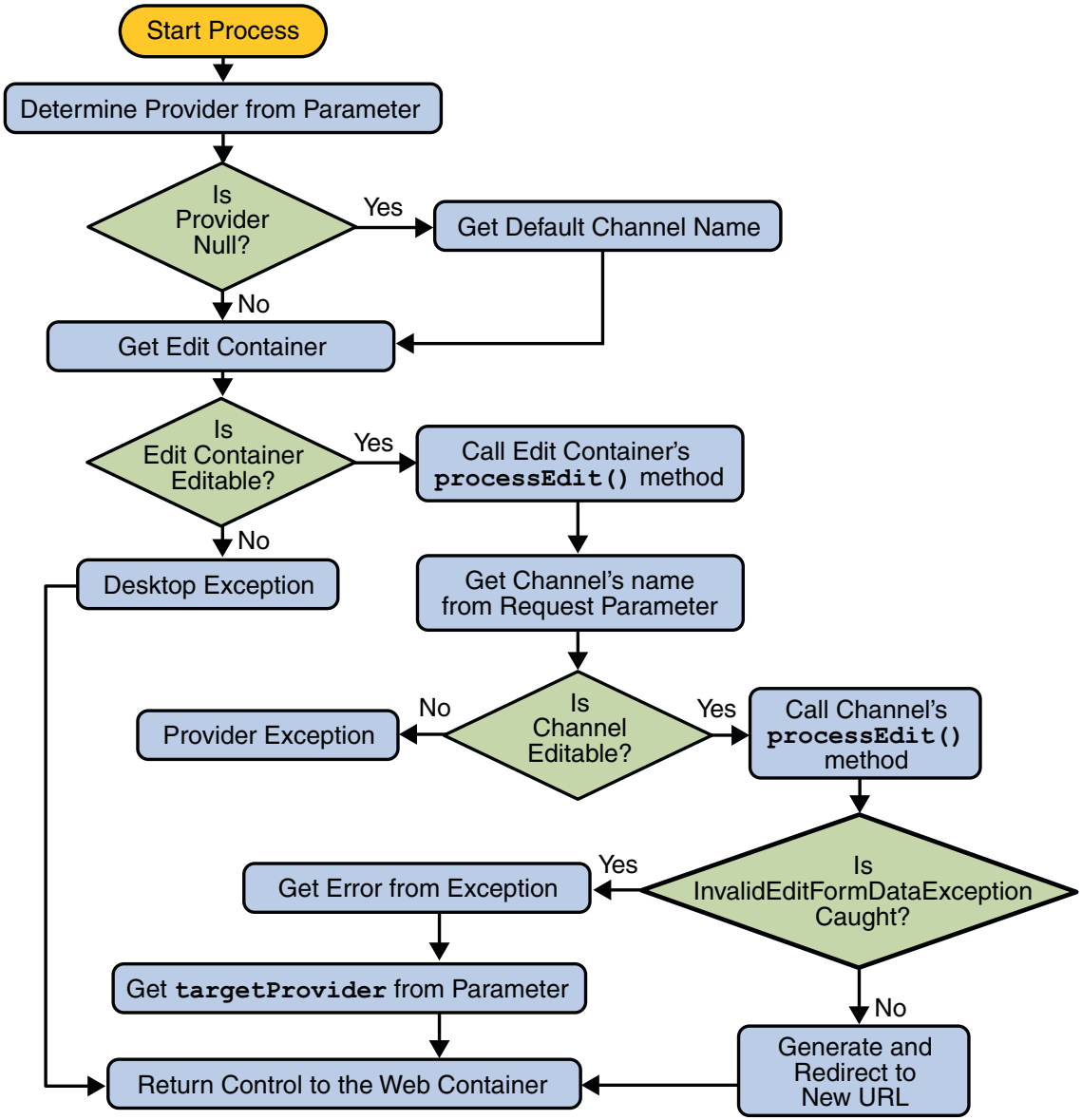

FIGURE 3–5 DesktopServlet process Action

Action logout

The logout action ends the user session. When the DesktopServlet receives a request where the action is logout, to perform the logout action, it takes the following parameters:

DesktopServletURL`/dt?action=logout`

When the DesktopServlet receives a request for the logout action, it redirects the browser to a URL defined by the Access Manager software `iplanet-am-platform-logout-url` attribute in the Platform service. By default, this attribute has the value `/amserver/logout`. But, If this is set to something that does not terminate the user's session, such as a static HTML page, then `/portal/dt?action=logout` will not terminate the user session.

◆ ◆ ◆ CHAPTER 4

Overview of the Leaf Providers

A portal provider is a generic connector to the resource in a channel and is the programmatic entity responsible for generating channels on the Desktop. A single provider can be used to create multiple channel instances. The leaf providers are tools for building channels for the portal.

This chapter contains the following:

Introduction to Leaf Providers

This section shows the relationship between the PAPI base class and the leaf providers discussed in this chapter. The JSPProvider and URLScraperProvider extend the ProfileProviderAdapter to include support for retrieving content from JSPs and URLs respectively. The XMLProvider extends the URLScraperProvider's functionality to scrape XML content from the named source and translate it into the markup language supported by the requesting client. The XMLProvider uses the URLScraperProvider's built-in functionality to fetch the XML contents from HTTP, HTTPs, and file URLs.

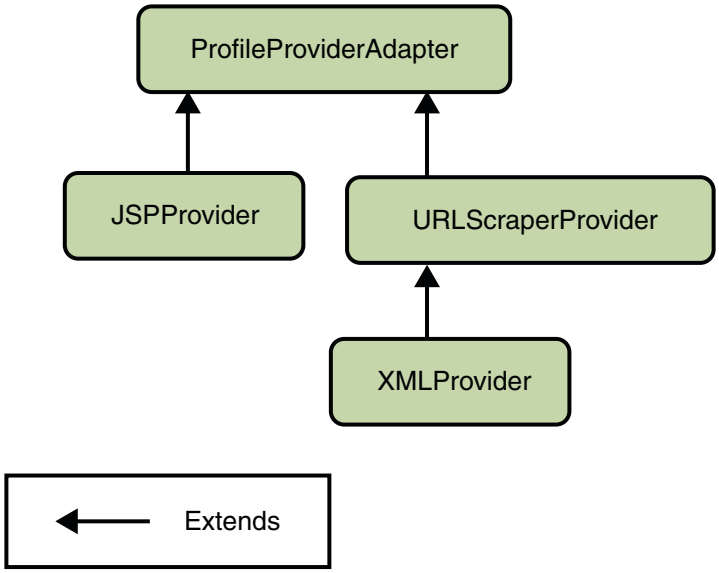

FIGURE 4–1 The Building Block Providers

JSPProvider

A JSPProvider is a content provider that can use JSP to create the content for a channel on the Desktop.

The step-by-step process of the JSPProvider invoking a JSP is:

1. The `JSPProvider.getContent()` method is called and the JSPProvider looks up the JSP to be executed. It calls a common private method to service the JSP.
2. The provider then creates request and response objects, copies information (such as cookies) from the Desktop request to those objects, initializes the session information, and checks to see if a servlet wrapper for the JSP already exists. If not, a servlet wrapper is created. The service method is then called on the wrapper.
3. The servlet wrapper service method loads the JSP. Before loading, it checks to see if the JSP class file is out of date with the class file. If the JSP class file is out of date with the class file, it invokes the compiler to compile the JSP into a class file and read the class file into memory and uses the class loader to define the new class.
4. Finally, the JSP servlet is instantiated and the `init()` method is called on the JSP.
5. Once the JSP is loaded, the JSP's service method is called by the servlet wrapper service method.
6. After the JSP's service method completes, the JSPProvider `getContent()` method extracts the response body from the processed JSP file and returns it as a StringBuffer.

URLScraperProvider

The URLScraperProvider can retrieve and display content from a given URL. The step-by-step description of how the URLScraperProvider retrieves and displays content from the specified URL.

1. Gets the `timeout` property for the provider.
2. Gets the `url` property for the provider.

   This is the URL source for the content to be fetched. The `url` property is fetched from the conditional property of display profile based on the user's clientType and locale, if defined. Otherwise the default value is returned. When scraping contents using URLScraperProvider, avoid the following technologies.

   - Frames and iFrames
   - Dynamic Hypertext Markup Language (DHTML)
   - Pages with extensive JavaScript™ technology that assumes the pages have the web browser to themselves
3. Gets the `urlScraperRulesetID` to be used by Rewriter as a string.

   This process ensures that any HTML links within the content are correct when included in the desktop. That is, when content included in the portal by using the URLScraperProvider often contains URLs to other documents, the Rewriter rewrites URLs in the scraped content by changing each link's relative links to absolute links thus ensuring that each link points back to the web site from which it was scraped. The Ruleset determines which tags in the HTML document contains URLs and must be rewritten. If the URLScraperProvider is scraping content that contains custom tags, extend the Ruleset to take these into account.
4. Determines if the `cookieName` can be forwarded.

   To determine, it checks the value of `cookiesToForwardAll` attribute value in the display profile. If this value is false, it checks the `cookiesToForwardList` attribute value in the display profile.
5. Determines if cookies from the request should be forwarded.

   The URLScraperProvider includes the functionality to forward cookies associated with the document intended for the client's web browser. Administrator's can control cookie forwarding on a per channel basis, forwarding some or all cookies to the user.
6. Gets the content by one of the following methods.

   - Retrieve content from the specified URL.

     That is, it sends an HTTP(s) request that contains information related to this request for content and gets an HTTP(s) response that allows the provider to influence the overall response for the Desktop page (besides generating the content).
   - Take the fully qualified path name of the file and returning StringBuffer containing the data from the specified file or null if file does not exist or cannot be read.

XMLProvider

The XMLProvider can be used to convert and display an XML file according to the specified style sheet.

The XMLProvider first determines if content is available for the requesting client. This is determined by checking on the availability of the XSL stylesheet for the requesting client via the File Lookup API.

The path to the XML content can be specified as HTTP, HTTPs, or file URL. The path to the XML source is stored as a value of the property `url`. If the path is HTTP or HTTPs, the provider uses the `super.getContent()` method to fetch the XML content. If the path is a file URL, the provider reads the file into a StringBuffer.

The value for XSL file name to use for transformation is stored as value of the string property `xslFileName`. The value for the attribute can be either the absolute path (such as `/export/home/xsl/abc.xsl`) or just the file name. If just the file name has been specified, the provider will use the File Lookup API to locate the XSL file. If the absolute path is specified, it takes precedence and file lookup will not be performed to locate the XSL file.

If both the XML and the XSL files are present, the XMLProvider does the transformation using the XSLT engine. The generated contents are displayed in the channel. In order to transform the XML file, the provider uses JAXP (from JWSDP 1.3).

The XMLProvider:

1. Takes the fully qualified path name of the XML file and returns the file, specified by the argument, as a StringBuffer.

   It returns null if the file does not exists or cannot be read.
2. Gets the XSL file. If just an XSL filename has been specified, the `getXSL()` method locates the XSL file using the File Lookup API.
3. Takes the XML and XSL files and does the transformation.
4. Gets the XML contents after converting the XML file according to the specified XSL stylesheet. In order to do this, it takes the HttpServletRequest and returns an HttpServletResponse with the XML file contents as a buffer.
5. Gets and displays the XML file contents in the channel using the `getContent()` method.

◆ ◆ ◆ CHAPTER 5

Overview of the Container Providers

Containers are simply channels that include in their output the content of one or more other channels. The container providers present containment metaphors for channel aggregation including single, table, tab, and framed.

This chapter discusses the out-of-the-box extensible set of container providers that implement various metaphors for aggregating content and contains the following sections:

The ContainerProvider Architecture

This section shows the relationship between the various interfaces and classes discussed in this chapter. The JSPProvider (see Chapter 4 for more information) and ContainerProviderAdapter classes extend ProfileProviderAdapter (see Chapter 2 for more information).

The ContainerProvider interface defines the interface for implementing a container provider. A container provider is a provider that generates its views primarily by being a client of other provider objects.

The ContainerProviderAdapter provides default implementations of the following methods in the ContainerProvider: `get/setSelectedChannels()`, `get/setAvailableChannels()`, `get/setWindowState()`, and `getSupportedWindowStates()`. For more information on all the methods in this class, see the Javadocs.

The ContainerProviderContext extends ProviderContext and adds container functionality to the ProviderContext interface. The ContainerProviderAdapter uses the ContainerProviderContext object as the persistent store.

The JSPContainerProviderAdapter extends the JSPProvider (see Chapter 4 for more information) and provides implementations of methods in the ContainerProvider interface to facilitate the execution of JSPs.

The JSPTabContainerProvider extends the JSPContainerProviderAdapter and provides default implementation for methods in the TabContainer interface. The JSPSingleContainerProvider and the JSPTableContainerProvider also extend the JSPContainerProviderAdapter.

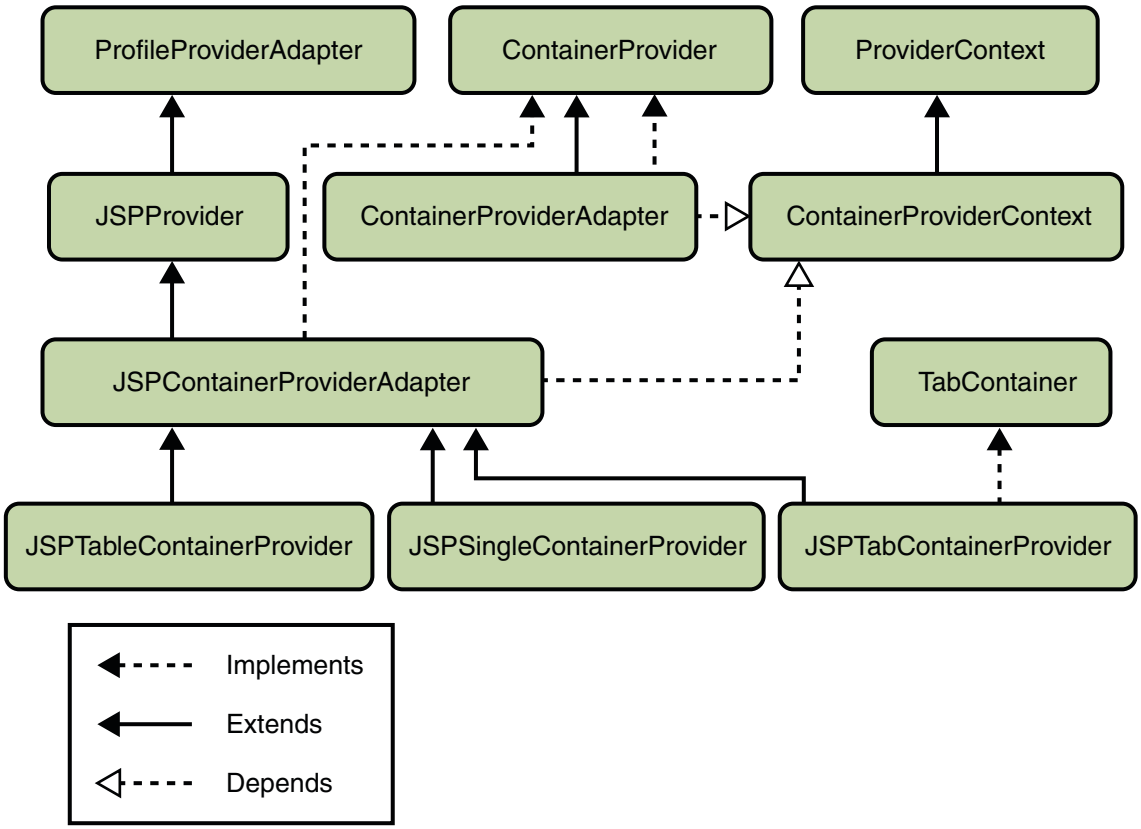

FIGURE 5–1 The ContainerProvider Architecture

Overview of the ContainerProviders

To support the container providers, the PAPI includes three APIs, the ContainerProvider interface, the ContainerProviderAdapter class, and the ContainerProviderContext interface. This section provides an overview of these three APIs.

ContainerProvider Interface

The ContainerProvider interface defines the interface for implementing a container provider and is the programmatic entity for generating container channels.

A container provider has a selected and available channels list, and allows getting and setting of these lists. Selected channels are those that are considered active on the Desktop. Available channels are those that are available to be activated on the Desktop. That is, selected channels are those that are available and selected for display on the Desktop by the user; available channels are those that are available for display on the Desktop, but not selected for display on the Desktop by the user.

The Desktop container providers (such as JSPSingleContainerProvider, JSPTableContainerProvider, and JSPTabContainerProvider) discussed in this chapter extend the JSPContainerProviderAdapter that implements the ContainerProvider interface.

The ContainerProvider includes interfaces to `get/setAvailableChannels` and `get/setSelectedChannels`. See the Javadocs for more information.

ContainerProviderContext Interface

The ContainerProviderContext interface provides an environment for container provider execution. More formally, this class adds container channel functionality to the ProviderContext interface (see Chapter 2 for more information). That is, where provider object can obtain access to a ProviderContext object, container providers obtain access to a ContainerProviderContext object.

A ContainerProviderContext object has a superset of the functionality exposed in ProviderContext. The additions are related to managing contained providers. For example, container channel functionality includes getting provider objects, fetching content, content caching, adding clients, creating channels and containers, and getting and setting selected and available channels list.

The ContainerProviderContext is the object to get the container channel properties from the store at run time. The ContainerProviderContext can also be used to `getProvider()` on another channel or container and `getContent()` on another channel or container.

ContainerProviderAdapter Class

The ContainerProviderAdapter class extends ProfileProviderAdapter. ContainerProviderAdapter can be used as the base class for any container provider (except for the JSP-based container providers, which can extend the JSPContainerProviderAdapter). For example, the sample template-based Desktop container providers are developed based on this class.

The ContainerProviderAdapter class provides default implementations of methods in the ContainerProvider interface implemented using a ContainerProviderContext object as the persistent store. This class also has the `getContainerProviderContext()` method, which gets the container provider context, in addition to the `get/setAvailableChannels` and `get/setSelectedChannels` methods.

JSPContainerProviderAdapter Class

To extend container functionality for JSP-based container providers, a JSPContainerProviderAdapter class is included with the software. This class extends the JSPProvider. The JSPContainerProviderAdapter can be used as the base class for any JSP-based ContainerProvider and is similar in functionality to the ContainerProviderAdapter. For example, the JSPSingleContainerProvider extends from this class.

The JSPContainerProviderAdapter class provides default implementations of methods in the ContainerProvider interface implemented using a ContainerProviderContext object as the persistent store and extends JSPProvider to facilitate the execution of JSPs. It includes interfaces to `get/setAvailableChannels()`, `get/setSelectedChannels()`, and `getContainerProviderContext()`.

Three sample extensions (namely "JSPSingleContainerProvider Class" on page 72, "JSPTableContainerProvider Class" on page 73, and "JSPTabContainerProvider Class" on page 75) to the JSPContainerProviderAdapter API are included.

JSPSingleContainerProvider Class

This section contains discusses about the JSPSingleContainerProvider in detail:

Introduction to JSPSingleContainerProvider

A single container wraps the content of a single channel. The single container enables a channel to take over an entire browser page. For example, this can be used to provide the front page or

can be used to display a single channel whose name is passed in the request parameter. Typically, the front page consists of some banners and the output of another channel. The purpose of the single channel is to allow these banners, menu bars, and the like to be wrapped around the content of the included channel.

Another purpose of the single container is that the decorational elements (such as banners and menu bars) which wrap around the channel can be easily changed without changing the channel itself.

Using the JSPSingleContainerProvider

The JSPSingleContainerProvider class extends JSPContainerProviderAdapter. A single container simply displays a single leaf channel or a container. It must be a JSP that wraps a container or leaf channel.

▼ The JSPSingleContainerProvider Class

1 **Gets the selected channel name.**

2 **Returns the selected channel name as a String.**

If more than one channel is defined, it displays the first channel in the list.

Some examples of single container include JSPContentContainer, JSPLayoutContainer, JSPEditContainer.

JSPTableContainerProvider Class

This section discusses about the JSPTableContainerProvider in detail:

Introduction to JSPTableContainerProvider

A table container aggregates the content of other channels into rows and columns. It can be thought of as a bucket for the content of other channels. The JSPTableContainerProvider class extends JSPContainerProviderAdapter.

The table container facilitates the aggregation of multiple channels into a single display. That is, the JSPTableContainerProvider aggregates channels into HTML rows and columns.

Using the JSPTableContainerProvider

A JSPTableContainerProvider provides methods that allow the associated JSP files to use these methods, and generate a view that arranges the contained channels to be displayed in a HTML table. By the nature of the container provider, the JSPTableContainer has a list of available channels and a list of selected channels.

Available channels are the channels that are available to be activated in the Desktop, and selected channels are the channels that are displayed in the Desktop. Selected and available channels list is defined in the table container channel definition in the display profile.

The TabContainer

A tab container aggregates the output of channels, providing a tabbed user interface to switch between them. A tab container's configuration can be modified at runtime to vary which leaf channel is displayed.

A tab container provider generates its views primarily by being a client of table container objects. The TabContainer displays one of its contained channels at a time. It allows containers to be arranged onto virtual pages. The container can then switch between these pages allowing them to be physically viewed one at a time.

It allows the user to switch logically separate row-column displays. From the container perspective, each page is a table container. The tab container then contains multiple table containers, one for each page. By default, each tab in a tab container corresponds to a table container. The tab container can contain any number of table, single, or tab containers theoretically. Having a tab container inside a tab container is not advisable.

A tab container provider is a container provider that has a selected and available channels list, and allows getting and setting of these lists. Selected channels are the table containers that are considered active on the Desktop. Available channels are those that are available to be activated on the Desktop.

The TabContainer interface defines the interface for implementing a TabContainerProvider. A TabContainerProvider must implement this interface. This interface contains methods to query information about a TabContainer and to set the properties of a TabContainer. See the Javadocs for more information on the methods in this interface.

UnmodifiableTab Interface

The UnmodifiableTab interface represents a tab that cannot be modified. This interface includes methods to:

- Get the name (`getName()`), the display name (`getDisplayname()`), the HTML encoded name (`getEncodedName()`), and description (`getDesc()`) of the tab as a string.

- Get the properties (`getTabMap()`) of the tab as a map.
- Determine whether or not the tab is removable (`isRemovable()`) and renamable (`isRenamable()`) by returning true or false.
- Determine if the tab was created by the user or was predefined for the user (`isPredefined()`) by returning true or false. A value of true indicates that the tab was predefined and a value of false indicates that the tab was created by the user.

ModifiableTab Interface

The ModifiableTab interface represents an instance of the tab that can be modified. It extends UnmodifiableTab and all modifiable tabs inherit the characteristics of the UnmodifiableTab. This interface provides methods to allow the user to reset the display name (`setDisplayname()`) and description (`setDesc()`) of the predefined tab.

JSPTabContainerProvider Class

The JSPTabContainerProvider includes the functionality to enable display of JSPs in the TabContainer environment. It provides methods that allow the associated JSP files to use these methods, and generate a view that aggregates the output of channels, providing a tabbed user interface to switch between them.

The JSPTabContainerProvider class extends JSPContainerProviderAdapter and implements the TabContainer interface.

◆ ◆ ◆ CHAPTER 6

Overview of Implementing a Custom Provider

This chapter contains the following sections:

Overview of Defining a Custom Provider

Before developing a custom provider, determine whether there is an existing Provider that can access the type of content you are targeting. For example, if you are trying to fetch content from a JSP page, you can create a channel based on the the pre-built JSP provider and configure it to access your JSP page. If none of the APIs can be used as is, determine whether or not the content can be presented by authoring a new provider by extending a provider base class (such as the ProviderAdapter or ProfileProviderAdapter).

The sections below explains how to develop a provider:

- "Defining Specific Requirements and Functionality" on page 78 that includes defining
  - Application Specific Requirements
  - Presentation Method
  - Provider-specific Properties
- "Developing the Provider Class File" on page 82
- "Creating XML Fragments for Display Profile" on page 85
- "Installing the Provider" on page 86

Defining Specific Requirements and Functionality

Before beginning to develop a provider, determine the application specific requirements, content source, and properties of the provider.

Application Specific Requirements

Determine the nature of the provider. Will this provider be a building block provider or will this be a content provider? The content providers are special purpose providers and the building block providers are general purpose providers. For example, the bookmark channel is special purpose, where XMLProvider is general purpose. That is, the XMLProvider can be used, in general, to connect to an XML source and translate it via XSLT to some markup language.

Determine if the content made available via this provider will be:

- Preset for the user by the administrator thus making it unmodifiable by the end-user. This will determine the logistics of the `isEditable()` method.
- Preconfigured for the user by the administrator, but editable by the end-user (determines the logistics of the `getEdit()` method). If editable, determine:
  - What will and what will not be editable. That is, determine the parameters that will be configurable and those that will not be editable.
  - The type of edit form to present - `EDIT_SUBSET` or `EDIT_COMPLETE`. The advantage of using `EDIT_SUBSET` is that the EditContainer provides a common look and feel for all providers that have `EDIT_SUBSET` edit type. Using `EDIT_COMPLETE` edit type provides more control over the overall look and feel for the edit page.

    This will determine the logistics of the `getEditType()` and the `processEdit()` methods.

  Determine the theme requirements for the content made available via this provider. A theme is a collection of visual elements, such as background color, font face, and so on, to be used when displaying the provider's content. The themes are stored as global properties in the organization level's display profile. Providers can use these properties in their templates or JSPs to achieve a global look and feel of the presentation. For use with JSP, theme properties can be accessed from a series of tag libraries. Non-JSP channels can access theme properties via the Theme class. See the Javadocs for more detail.

Presentation Method

Determine whether the provider will fetch content from template files or JSPs. This will determine the logistics of the `getTemplate()` method.

Channel template files are stored in a directory based on the name of the channel, or the name of the provider that is used by the channel. The channel directory for the provider is created under the template root directory. By default, this will be:

PortalServer-DataDir/portals/*portal-ID*/desktop/*desktoptype*/*channeldirectory*/*templatefiles*

The JSPs and templates for desktop type `default` are stored under *PortalServer-DataDir*/portals/*portal-ID*/desktop/default directory. Storing the files in the *PortalServer-DataDir*/portals/*portal-ID*/desktop/default directory allows the files to be shared. That is, when the Desktop service uses the comma-separated string (as an ordered Desktop type list) in the Desktop type attribute to lookup, it starts at the first element in the list and each element represents a subdirectory under the Desktop template base directory. If a template is not found in the first directory (or the first string in the desktop type attribute), then it proceeds to the next one in the list. This continues until the item is found (or not), for all Desktop type elements in the list. If the default directory is not included in the list, it will be added at the end of the list implicitly.

The directory search order for the template and JSP files is as follows:

desktoptype_locale/channelname/clientPath
desktoptype_locale/provider/clientPath
desktoptype_locale/channelname
desktoptype_locale/provider
desktoptype_locale/clientPath
desktoptype_locale
desktoptype/channelname/clientPath
desktoptype/provider/clientPath
desktoptype/channelname
desktoptype/provider
desktoptype
default_*locale/channelname/clientPath*
default_*locale/provider/clientPath*
default_*locale/channelname*
default_*locale/provider*
default_*locale*
default/*channelname/clientPath*
default/*provider/clientPath*
default/*channelname*
default/*provider*
default/*clientPath*
default
templateroot

Where

- *desktoptype* is the value of the `desktopType` property
- *locale* is the user's locale
- *channelname* is the name of the channel
- *provider* is the provider name
- *clientPath* is an optional file-path containing client-specific templates
- *templateroot* is, by default, *PortalServer-DataDir*/portals/*portal-ID*/desktop

If there is no *clientPath* specified, then the directory search order is as follows:

desktoptype_locale/channelname
desktoptype_locale/provider
desktoptype_locale
desktoptype/channelname
desktoptype/provider
desktoptype
`default_`*locale/channelname*
`default_`*locale/provider*
`default_`*locale*
`default/`*channelname*
`default/`*provider*
`default`
templateroot

Note – If the channel display profile is defined inside a container display profile definition, the lookup for the channel will be *desktoptype/containername/channelname/clientPath*.

Provider-specific Properties

In the display profile, properties contain configuration information for the provider and the provider's channels. A provider can define any number of specific properties. See "Properties in Display Profile" on page 80 and the *Sun Java System Portal Server 7.1 Technical Reference* for more information.

Also, a provider can have more than one properties file, which is actually a Java resource bundle file that enables localization of on-screen images or text. See "Properties in Resource bundle" on page 81 for more information.

Properties in Display Profile

The display profile document defines a Provider. The definition of a Provider associates the Provider with the class file that implements its behavior. A Provider definition includes a number of properties, mandatory and provider-specific.

Provider specific properties are defined in the display profile. Provider specific properties are defined in the display profile within the `<Properties></Properties>` tag for the provider. The following is the structure of the properties in the display profile:

```
<Properties>
    <Collection>
        ...subset of property definitions
    </Collection>
        <Integer>...</Integer>
```

```
        <String>...</String>
        <Boolean>...</Boolean>
</Properties>
```

Typically, the custom provider will not modify the required properties. When properties are updated, they are set in the user's display profile by the administrator via the administration console. However, values can only be set on existing properties.

If the provider class does not extend the ProfileProviderAdapter and implements the Provider interface, then the `get*Property()` methods cannot be used directly, and the display profile properties are not accessible. If the provider class implements the Provider interface directly, then no properties are required. Properties must be hardcoded return values or can come from another source such as a database. However, some provider properties are required if the provider implementation extends ProviderAdapter or ProfileProviderAdapter.

The following is a list of the mandatory properties for a provider. The mandatory properties described below are for a provider extending the ProviderAdapter or the ProfileProviderAdapter classes. If you are implementing the Provider interface directly, you can avoid the use of the display profile and the required properties outlined below.

`title`	Title of the Provider
`description`	Description of the Provider
`refreshTime`	Time interval for cached content
`isEditable`	Can the Provider be edited
`editType`	Type of edit form to use
`width`	Width of the Provider
`helpURL`	URL of a help page for this Provider

See *Technical Reference Guide* for detailed information on the Display Profile properties.

Channel definitions that use a provider can overwrite the properties of that provider. At runtime, channel property values can be affected by the display profile merge process. See *Technical Reference Guide* for detailed information on the display profile merge process.

Properties in Resource bundle

Strings defined in the provider's resource bundle properties file are displayed on the user's Desktop and they are not the channel's properties (such as the channel title). Strings are defined in the resource bundle to enable localization.

Typically, resource bundle includes properties that do not need to be customized by the administrator or the end-user. For example, properties that need to be localized, are read-only

at runtime, and properties that do not change per channel should be included in a resource bundle. Properties that need to be localized, are settable, and properties that change per channel should be in the display profile.

By default, a provider's resource bundle properties file is stored in *PortalServer-base*/web-src/WEB-INF/classes directory. Note that you can change the file here, but you must re-deploy the portal web application for the changes to take effect.

Resource bundles that are needed by the custom providers should be copied into the provider class base directory (see "Provider Class File" on page 83 for more information.) Resource bundles should be placed as individual files and cannot exist inside the JAR file under the provider class base directory.

Resource bundles are given a base name that equals to the name of the display profile provider definition that they are associated with. Typically, this is the class name of the Java class file that implements the provider.

To get the name of the resource bundle in your custom provider, call `ProviderContext.getProviderName()`. This value can then be used as the `baseName` argument to `ResourceBundle.getBundle()`. If you are extending ProviderAdapter, simply call `getResourceBundle()`.

Developing the Provider Class File

This section includes information on the requirements for developing a provider's class file. It contains:

Provider Class File Location

The provider classes must be deployed in the provider class base directory specified in the file *PortalServer-DataDir*/portals/*portal-ID*/config/desktopconfig.properties. By default, the provider class base directory is *PortalServer-DataDir*/portals/*portal-ID*/desktop/classes.

For hot deployment, the JAR file has to be dropped in the directory mentioned in `providerClassBaseDir` variable in the `desktopconfig.properties` file.

Provider Class File

The custom provider classes and all other custom classes that the provider classes reference can be bundled into a JAR file and placed in the provider class base directory. They can also exist as `.class` files in the provider class base directory.

If a class exists in a file and is also present in a JAR file under the provider class base directory, the class existing as a file under the provider class base directory is given the first preference. Once the class is found as a file, no attempt is made to check for its existence in a JAR file.

If a class file exists in two JAR files under the provider class base directory, the class is picked up from one of them at runtime. Do not duplicate classes in multiple JAR files.

When compiling the class file, the Servlet 2.2 API (`servlet.jar`) and PAPI (*PortalServer-base*`/sdk/desktop/desktopsdk.jar`) are required in the `CLASSPATH`. That is, when compiling a `*`*Provider*`.java` file, you must type:

```
javac -d PortalServer-DataDir/portals/portal-ID/desktop/classes -classpath PortalServer-base
/sdk/desktop/desktopsdk.jar:AccessManager-base/lib/servlet.jar *Provider.java
```

Provider Class Loader

To support hot deployment of Provider classes the Portal Server has a Provider classloader implementation that can pickup changes to custom classes in the Provider Classloader directory. The provider class loader in the Portal Server software is used to load the classes.

A Provider Classloader instance loads all custom provider classes and resource bundles. When a class in the Provider Classloader directory changes, the Provider classloader and all the classes it loaded are discarded; a new Provider Classloader object is created and this one reloads all the custom classes.

In order to upload a new version of a custom provider class without restarting the server, if the original class existed in a JAR file, rebuild the JAR with the new version of the class file and copy the JAR into the provider class base directory. Or, Just copy the new version of the class as a file into the provider class base directory.

In order to reload a property file, that is being used by a custom provider, with a modified property inside a resource bundle in the provider class base directory without restarting the server, change the last modified time on the custom provider class. If the custom provider class exists:

- In a JAR file, use the command `touch` *JAR-file-name*`.jar`
- As a file, use the command `touch` *class-file-name*

The presence of the provider class loader has an impact on the design of a multi-channel application in the Desktop. When an object reference is cast from one type to another, not only does the class of the variable need to match the class of the object, the two associated class loaders have to be the same too. This constraint can show up when objects are being passed between different channels in the Desktop.

For example, consider a portal application that uses two providers, one for a container and the other for several leaf channels. Call these providers HeadlineContainerProvider and HeadlineProvider. The container provider performs some common processing for the leaf channels, and then the leaf channels access the container object to get the results of that processing. One way to implement this would be to have the HeadlineContainerProvider put a reference to itself into the request object. For example, in the `HeadlineContainerProvider.getContent()`, do:

```
request.setAttribute("HeadlineContainer", this);
```

Then, in the `HeadlineProvider.getContent()` method, do:

```
HeadlineContainerProvider hcp = (HeadlineContainerProvider)
request.getAttribute("HeadlineContainer");
```

This example will compile, but when the desktop loads and runs this code, the HeadlineProvider will cause a ClassCastException on this second statement. The reason is that the HeadlinerContainerProvider object is loaded by one provider class loader object and the HeadlineProvider object is loaded by a different class loader object. The class loader for HeadlineProvider also loads HeadlineContainerProvider because HeadlineProvider references HeadlineContainerProvider. So we have:

- The container1 of type HeadlineContainerProvider loaded by loader1
- The channel1 of type HeadlinerProvider loaded by loader2

The HeadlineContainerProvider class is also loaded by loader2 because of the reference to it in the HeadlineProvider code. The cast is from the object container1 (loaded by loader1) to the variable `hcp` (of type HeadlineContainerProvider loaded by loader2). Thus the cast fails because the class loaders are not the same.

To understand a case where the classes could actually be different, consider the case where container1 is loaded at time 0, and channel1 is loaded at time 10, but the HeadlineContainerProvider class files was updated on disk at time 5. This means that channel1 will be running with a different version of the HeadlineContainerProvider class than that being used for container1. The case must fail because the class versions are different.

To implement this type of application, an alternate solution is to pass an object as a request or session property that has a class that is not in the *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes` directory, such as String, Set, or

Hashmap. These classes are never reloaded except when the server restarts. The data that you store in that object becomes a contract between different versions of the classes that use that object.

Creating XML Fragments for Display Profile

A provider must be defined in the display profile before it can be used. This definition associates the provider with its class file implementation and includes the provider's properties. That is, the provider's display profile fragment should include default values for all the properties that are used in the *provider*`.java` file. For example, if the *provider*`.java` file contains `getStringProperty("color")`, the provider's display profile fragment should include a default value for `color`.

The provider's XML fragment in the display profile must adhere the following structure:

```
<DisplayProfile>
    <Providers>
        <Provider name="
            providername"
                class="
            provider class name">
                <Properties>...</Properties>
        </Provider>
    </Providers>
</DisplayProfile>
```

The provider definition is the template that decides the properties for the provider's channels. However, the display profile definition for the channel ultimately decides the values for the channel's properties. When a channel sets a property, the property is set in the channel and not in the provider's properties. When defining channel properties in the display profile, include only those properties where the provider defaults are not applicable.

The provider's channel XML fragment in the display profile must adhere the following structure:

```
<DisplayProfile>
    <Channels>
        <Channel name="
            channelname" provider="
            providername">
                <Properties>...</Properties>
        </Channel>
    </Channels>
</DisplayProfile>
```

The channel definition in the display profile need not have the `<Container>` tag unless the channel is defined within a container. Channel definitions in the display profile can be encapsulated inside a container. The advantage of this is that it provides name scoping so that channels with the same name do not collide.

Unless a channel is referenced directly from the DesktopServlet URL, in which case it need not be encapsulated inside a container, you must reference a channel from some container in order to see it.

The container within which the channel will operate must adhere the following structure:

```
<DisplayProfile>
    <Channels>
        <Container>
            <Properties>...</Properties>
            <Available>...</Available>
            <Selected>...</Selected>
            <Channel>...</Channel>
        </Container>
    </Channels>
</DisplayProfile>
```

To upload display profile changes, use the `psadmin add-display-profile` or `psadmin modify-display-profile` sub command. See *Sun Java System Portal Server 7.1 Command Line Reference* for more information on the `psadmin` command and see the Portal Server 7.1 administration console online help for more information on administering the display profile.

Installing the Provider

This section provides information on:

Installing Manually

▼ To install the provider manually

1 **Compile the provider class file and copy the file to the provider class base directory which, by default, is** *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes`**. Or copy the file to the location specified in the** *PortalServer-DataDir*`/portals/`*portal-ID*`/config/desktopconfig.properties` **file.**

2. **Copy the resource bundle files, if any, to** *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes` **directory.**

3. **Develop the JSPs and templates for the provider and copy the files to the template root directory for the provider which, by default, is** *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype*`/`*providerName***.**

4. **Develop and upload the display profile XML fragments for the provider, the provider's channels, and the container within which the channels will operate, if any.**

Transporting Using the `psadmin` Utility

Utilize the `psadmin export/import -t provider` utility for deploying channels and providers. The `psadmin export/import -t provider` command enables you to transfer or move providers or channels from one the Portal Server software host to another. The `psadmin export/import -t provider` utility creates a specialized packaging mechanism called a `.par` file for transport of channels and providers into and out of the server.

A `.par` file is an extended form of the `.jar` file format, with added:

- MANIFEST information to carry the deployment information. The special records are called auto-extract operations
- XML document intended for integration into the Portal Server software display profile on the target server

The use of `.par` files is optional. You can just as well copy all files into their correct locations. The `par` utility makes deployment easier if you have to install the files on multiple servers.

To create a `.par` file from scratch, use the `export` sub command of the `psadmin` utility.

To export an existing provider, use the `export` sub command of the `psadmin` utility.

For example, if you have the provider associated files in the development environment ready for deployment into the production environment, use the following command to build the channel and provider into a PAR file which can then be deployed on the system.

`psadmin export -u` *dn_amadmin* `-f` *passwordfile* `-t provider -p` *portal-ID* `-d|-g -x` *exportfile provider*`.par`

To deploy the `.par` file into the system, use the following command:

`psadmin import -u` *dn_amadmin* `-f` *passwordfile* `-p` *portal-ID dn* `-O`
`'"provider=`*providername*`|channel=`*channelname*`","provider=`*providername*`|channel=`*channelnam*

Note – All providers that ship with the Portal Server software are in the service-level Desktop display profile. There is no command to extract them all.

See *Sun Java System Portal Server 7.1 Command Line Reference* for more information on the `psadmin` utility.

◆ ◆ ◆ CHAPTER 7

Extending the Base Providers

This chapter includes instructions for creating a custom provider by:

Examples are provided for developing a sample HelloWorldProvider that will display a message ("Hello World"). This is to illustrate the fundamentals of how the Provider interface works first, followed by examples of how ProviderAdapter and ProfileProviderAdapter classes make the implementation easier. Use the instructions (provided with the sample HelloWorldProvider) for developing your custom provider by extending any one of these APIs.

Implementing the Provider Interface

The Provider interface defines the interface for implementing the provider component of a channel. If you want to directly implement the Provider interface, code should be written to implement all of the methods in this interface. You will not have access to the ProviderContext and the properties in the display profile cannot be accessed. If you are implementing this interface, it's up to the provider implementation to get the properties.

For more information on this interface, see the Javadocs at `http://`*`hostname`*`.`*`port`*`/portal/javadocs`.

▼ To create a custom provider by implementing the Provider interface

This section provides the instructions for creating a custom provider by implementing the Provider interface. The process described here includes creating a sample HelloWorldProvider that prints "Hello World!" on the provider's channel.

1 Create a new Java class which implements the Provider interface.

For the sample HelloWorldProvider, create the class file as shown below.

`HelloWorldProviderP.java` File

```
package custom;

import java.net.URL;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.sun.portal.providers.Provider;
import com.sun.portal.providers.ProviderException;
import com.sun.portal.providers.UnknownEditTypeException;

public class HelloWorldProviderP implements Provider {
    public void init(
        String name, HttpServletRequest req
    ) throws ProviderException { }

    public StringBuffer getContent(
        HttpServletRequest request, HttpServletResponse response
        ) throws ProviderException {
        return new StringBuffer("Hello World!");
    }

    public java.lang.StringBuffer getContent(java.util.Map m)
        throws ProviderException {
        return new StringBuffer("Hello, world!");
    }

    public StringBuffer getEdit(HttpServletRequest request, HttpServletResponse
        response) throws ProviderException {
        // Get the edit page and return it is as a StringBuffer.
        return null;
    }

    public java.lang.StringBuffer getEdit(java.util.Map m) throws
        ProviderException {
        return null;
    }

    public int getEditType() throws UnknownEditTypeException {
        return 0;
    }

    public URL processEdit(HttpServletRequest request,
        HttpServletResponse response) throws ProviderException {
        return null;
```

```
    }

    public java.net.URL processEdit(java.util.Map m) throws ProviderException {
        return null;
    }

    public boolean isEditable() throws ProviderException {
        return false;
    }

    public boolean isPresentable() {
        return true;
    }

    public boolean isPresentable(HttpServletRequest req) {
        return true;
    }

    public java.lang.String getTitle() throws ProviderException {
        return "HelloWorld Channel";
    }

    public java.lang.String getName() {
        return "HelloWorld!";
    }

    public java.lang.String getDescription() throws ProviderException {
        return "This is a sample HelloWorld channel";
    }

    public java.net.URL getHelp(javax.servlet.http.HttpServletRequest req)
        throws ProviderException {
        return null;
    }

    public long getRefreshTime() throws ProviderException {
        return 0;
    }

    public int getWidth() throws ProviderException {
        return 0;
    }
}
```

2 Compile the class and put it in the user defined class directory.

The default directory for the class file is *PortalServer-DataDir*/portals/*portal-ID*/desktop/classes. To compile the HelloWorldProviderP.java file, type:

```
javac -d PortalServer-DataDir/portals/portal-ID/desktop/classes -classpath PortalServer-base
/sdk/desktop/desktopsdk.jar:AccessManager-base
/lib/servlet.jar HelloWorldProviderP.java
```

3 Define the new provider and channel definition in a temporary XML file.

The sample HelloWorldProvider provider and channel definitions are in the HelloProviderP.xml and HelloChannelP.xml files.

HelloProviderP.xml File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Provider name="HelloWorldProviderP" class="custom.HelloWorldProviderP">
    <Properties>
        <String name="title" value="Hello World Channel"/>
        <String name="description">This is a test of the hello world provider</String>
        <String name="width" value="thin"/>
    </Properties>
</Provider>
```

HelloChannelP.xml File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Channel name="HelloWorldP" provider="HelloWorldProviderP">
    <Properties>
        <String name="message" value="Hello World Test!!! - non-localized "/>
        <Locale language="en" country="US">
        <String name="message" value="Hello World -
            I am speaking English in the United States!!!"/>
        </Locale>
    </Properties>
</Channel>
```

4 Upload the provider and channel XML fragments using the psadmin add-display-profile command.

For the sample HelloWorldProvider, upload the HelloProviderP.xml and HelloChannelP.xml XML fragments using the psadmin add-display-profile command. See the *Sun Java System Portal Server 7.1 Command Line Reference* for more information on psadmin add-display-profile command.

5 **Specify the URL in your browser to access the channel.**

`http://`*hostname*`:`*port*`/portal/dt?provider=HelloWorldP`

Extending the ProviderAdapter Class

As the ProviderAdapter class has implemented many of the methods of the Provider interface, the custom provider can extend ProviderAdapter and can override some of the methods or define any new methods. The advantage of extending this class versus implementing the Provider interface directly is that you will maintain forward compatibility as additions are made to the PAPI. Existing code will by default call the methods in this class. Also, the ProviderAdapter contains default implementations of methods in the Provider interface that you can use.

▼ To create a custom provider by extending the ProviderAdapter class

This section provides the instructions for creating a custom provider by extending the ProviderAdapter class. The process described here includes creating a sample HelloWorldProvider that prints "Hello World!" on the provider's channel.

1 **Create a new java class which extends the ProviderAdapter class.**

For the sample HelloWorldProvider, create the class file as shown here.

`HelloWorldProviderPA.java` File

```
package custom;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.sun.portal.providers.ProviderAdapter;
import com.sun.portal.providers.ProviderException;

public class HelloWorldProviderPA extends ProviderAdapter {
    public java.lang.StringBuffer getContent (HttpServletRequest request,
        HttpServletResponse response) throws ProviderException  {
        return new StringBuffer("Hello, world!");
    }
}
```

2 Compile the class and put it in the user defined class directory.

The default directory for the class file is *PortalServer-DataDir*/portals/*portal-ID*/desktop/classes. To compile the HelloWorldProviderPA.java file, type:

```
javac -d PortalServer-DataDir/portals/portal-ID/desktop/classes -classpath PortalServer-base
/sdk/desktop/desktopsdk.jar:AccessManager-base/lib/servlet.jar HelloWorldProviderPA.java
```

3 Define the new provider and provider's channel definition in a temporary XML file.

The sample HelloWorldProvider XML fragment for the provider is in the HelloProviderPA.xml file.

HelloProviderPA.xml File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Provider name="HelloWorldProviderPA" class="custom.HelloWorldProviderPA">
    <Properties>
        <String name="title" value="*** TITLE ***"/>
        <String name="description" value="*** DESCRIPTION ***"/>
        <String name="refreshTime" value="0"/>
        <Boolean name="isEditable" value="true"/>
        <String name="editType" value="edit_subset"/>
        <String name="width" value="thin" advanced="true"/>
        <ConditionalProperties condition="locale" value="en">
            <String name="helpURL" value="desktop/HelloWorld.htm" advanced="true"/>
        </ConditionalProperties>
        <String name="helpURL" value="desktop/HelloWorld.htm" advanced="true"/>
        <String name="fontFace1" value="Sans-serif"/>
        <String name="productName" value="Sun Java System Portal Server"/>
        <Collection name="aList">
            <String value="i'm aList"/>
        </Collection>
        <String name="message" value="Sample Hello World Provider"/>
    </Properties>
</Provider>
```

4 Upload the provider and channel XML fragments using the psadmin add-display-profile command.

For the sample HelloWorldProvider, upload the HelloProviderPA.xml XML fragments using the psadmin add-display-profile command. See the *Sun Java System Portal Server 7.1 Command Line Reference* for more information on psadmin add-display-profile command.

5 Include the new channel in one of the existing containers.

The sample HelloWorldProvider will be displayed only in a Table Desktop layout. To add the channel in the JSPTableContainer from the administration console, follow instructions in the Portal Server administation console online help. To add the channel in the JSPTableContainer manually:

a. Add the HelloWorldProvider channel XML fragment (in the `HelloChannelPA.xml` file) for the JSPTableContainer.

`HelloChannelPA.xml` File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">
<Container name="JSPTableContainer" provider="JSPTableContainerProvider">
<Properties>
    <Collection name="Personal Channels">
        <String value="HelloWorldPA"/>
    </Collection>
    <Collection name="channelsRow">
        <String name="HelloWorldPA" value="3"/>
    </Collection>
    <Collection name="channelsColumn">
        <String name="HelloWorldPA" value="3"/>
    </Collection>
</Properties>
<Available>
    <Reference value="HelloWorldPA"/>
</Available>
<Selected>
    <Reference value="HelloWorldPA"/>
</Selected>
<Channels>
    <Channel name="HelloWorldPA" provider="HelloWorldProviderPA">
        <Properties>
            <String name="message" value="Hello World Test!!! - non-localized "/>
        </Properties>
    </Channel>
</Channels>
</Container>
```

b. Use the `psadmin modify-display-profile` sub command to add the changes to the container.

If you do not specify a parent object with the `-p` option, the channel is added at the root level.

6 Specify the URL in your browser to access the HelloWorld channel inside the JSPTableContainer.

`http://`*hostname*`:`*port*`/portal/dt?provider=JSPTableContainer/HelloWorldPA`

Extending the ProfileProviderAdapter Class

ProfileProviderAdapter has the default implementation of the Provider interface and extends the ProviderAdapter class. It includes some convenient methods that allow the providers to get the channel data from the ProviderContext object. Developers who wish to create a new provider can take advantage of this, by extending the ProfileProviderAdapter class, and create their customized provider class.

▼ To Create a Provider by Extending the ProfileProviderAdapter Class

This section provides the instructions for creating a custom provider by extending the ProfileProviderAdapter class. The sample HelloWorldProvider reads a string property from the user's display profile and allows the user to edit the string. This sample also includes an example about using the resource bundle.

1 Create a new Java class which extends the ProfileProviderAdapter class.

For the sample HelloWorldProvider, create the class file.

`HelloWorldProviderPPA1.java File`

```
package custom;

import java.net.URL;
import java.util.Hashtable;
import java.util.ResourceBundle;
import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.sun.portal.providers.ProfileProviderAdapter;
import com.sun.portal.providers.ProviderException;

public class HelloWorldProviderPPA1 extends ProfileProviderAdapter {
    private ResourceBundle bundle = null;

    // Implement the getContent method
    public StringBuffer getContent(
        HttpServletRequest req, HttpServletResponse res
    ) throws ProviderException {
        Hashtable<String,Object> tags = new Hashtable<String,Object>();
        if (bundle == null) {
            bundle = getResourceBundle();
        }
```

```
            tags.put("welcome", bundle.getString("welcome"));
            tags.put("message", getStringProperty("message", true));
            tags.put("properties", getMapProperty("aMap", true));
            StringBuffer b = getTemplate("content.template", tags);
            return b;
        }

        // Implement the getEdit method
        public StringBuffer getEdit(
            HttpServletRequest req, HttpServletResponse res
        ) throws ProviderException {
                Hashtable<String,String> tags = new Hashtable<String,String>();
                tags.put("message", getStringProperty("message"));
                tags.put("properties", "");
                StringBuffer b = getTemplate("edit.template", tags);
                return b;
        }

        // Implement the processEdit method
        public URL processEdit (
            HttpServletRequest req, HttpServletResponse res
        ) throws ProviderException {
            String message = req.getParameter("message");
            if(message != null) {
                if (!message.equals(getStringProperty("message"))) {
                    // Set the new message
                    setStringProperty("message",message);
                    return null;
                }
            } else {
                    setStringProperty("message","");
                }
            return null;
        }
    }
```

2 Compile the class and put it in the user defined class directory.

The default directory for the class file is *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes`. To compile the `HelloWorldProviderPPA1.java` file, type:

`javac -d` *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes -classpath` *PortalServer-base*`/sdk/desktop/desktopsdk.jar:`*AccessManager-base*`/lib/servlet.jar HelloWorldProviderPPA1.java`

3 Define the new provider and provider's channel definition in a temporary XML file.

Following is the sample HelloWorldProvider XML fragment for the provider in the `HelloProviderPPA1.xml` file.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Provider name="HelloWorldProviderPPA1" class="custom.HelloWorldProviderPPA1">
    <Properties>
        <String name="title" value="*** TITLE ***"/>
        <String name="description" value="*** DESCRIPTION ***"/>
        <String name="refreshTime" value="0"/>
        <Boolean name="isEditable" value="true"/>
        <String name="editType" value="edit_subset"/>
        <Collection name="aMap">
            <String name="title" value="Ramag"/>
            <String name="desc" value="My Map"/>
            <Boolean name="removable" value="true"/>
            <Boolean name="renamable" value="true"/>
        </Collection>
        <String name="message" value="Hello World Test Provider"/>
    </Properties>
</Provider>
```

Following is the sample HelloWorldProvider XML fragment for the channel in the `HelloChannelPPA1.xml` file.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Channel name="HelloWorldPPA1" provider="HelloWorldProviderPPA1">
    <Properties>
        <String name="title" value="Hello World Channel"/>
        <String name="description">This is a test of the hello
        world provider</String>
        <String name="width" value="thin"/>
        <String name="message" value="Hello World Test!!! - non-localized "/>
        <Locale language="en" country="US">
            <String name="message" value="Hello World - I am speaking
            English in the United States!!!"/>
        </Locale>
    </Properties>
</Channel>
```

4 Upload the provider and channel XML fragments using the `psadmin add-display-profile` sub command.

For the sample HelloWorldProvider, upload the `HelloProviderPPA1.xml` file and `HelloChannelPPA1.xml` file XML fragments using the `psadmin add-display-profile` sub

command. See the *Sun Java System Portal Server 7.1 Command Line Reference* for more information on the `psadmin add-display-profile` sub command.

5 Include the provider's channel in one of the existing containers.

The sample HelloWorldProvider will be displayed in a Table Desktop layout. To add the channel in the JSPTableContainer from the administration console, follow instructions in the Portal Server administration console online help. To add the channel in the JSPTableContainer manually:

a. Create the HelloWorldProvider channel XML fragment (in the `HelloContainerPPA1.xml` file) for the JSPTableContainer as shown in below.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">
 <Container name="JSPTableContainer" provider="JSPTableContainerProvider">
        <Properties>
             <Collection name="Personal Channels">
                <String value="HelloWorldPPA1"/>
            </Collection>
            <Collection name="channelsRow">
                <String name="HelloWorldPPA1" value="3"/>
            </Collection>
        </Properties>
         <Available>
          <Reference value="HelloWorldPPA1"/>
         </Available>
         <Selected>
             <Reference value="HelloWorldPPA1"/>
         </Selected>
         <Channels>
         </Channels>
 </Container>
```

b. Use the `psadmin modify-display-profile` sub command to add the channel to a container.

If you do not specify a parent object with the `-p` option, the channel is added at the root level.

6 Create the template files if the new provider requires template files.

The sample HelloWorldProvider requires `content.template` and `edit.template` files.

a. Create the `content.template` file for the HelloWorldProvider.

The contents of `content.template` file is shown below:.

```
<html>
<head></head>
<body bgcolor="white">
[tag:welcome]<br><br>
```

```
[tag:message]<br><br>
[tag:properties]
</body>
</html>
```

b. Create the `edit.template` file for the HelloWorldProvider.

The contents of `edit.template` file is shown below:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
    <HEAD>
        [tag:noCache]
        <TITLE>[tag:productName]</TITLE>
    </HEAD>
    <BODY BGCOLOR="#FFFFFF">
        <FONT SIZE=+0 FACE="[tag:fontFace1]">
        <LINK REL="stylesheet" TYPE="text/css" HREF="
            [surl:/desktop/css/style.css]" TITLE="fonts">
        <CENTER>

        [tag:bulletColor]
        [tag:banner]

        <FORM ACTION="dt" NAME="edit_form" METHOD=POST
            ENCTYPE="application/x-www-form-urlencoded">
        <INPUT TYPE=HIDDEN NAME="action" SIZE=-1 VALUE="process">
        <INPUT TYPE=HIDDEN NAME="provider" SIZE=-1 VALUE="[tag:providerName]">

        [tag:inlineError]

        <TABLE BORDER=0 CELLPADDING=0 CELLSPACING=3 WIDTH="100%">
           <TR>
             <TD WIDTH="100%" VALIGN=TOP>
                 <CENTER>
                   <BR>
                   <FONT SIZE="+2" FACE="[tag:fontFace1]">
                   <B>
                   Edit [tag:title]</B></FONT>
                   <BR>
                   [tag:contentOptions]
                 </CENTER>
             </TD>
           </TR>
        </TABLE>

        <BR>

        <FONT SIZE=+0 FACE="[tag:fontFace1]">
        <INPUT TYPE=SUBMIT NAME="Submit" VALUE="Finished" CLASS="button">
```

```
            <INPUT TYPE=BUTTON OnClick="location='[tag:desktop_url]'"
                VALUE="Cancel" CLASS="button">
            </font>

            <br>

            <P>
            </FORM>
            <BR>
            [tag:menubar]
            </font>
        </BODY>
    </HTML>
```

7 Create the channel directory under the template root directory.

By default, the template root directory is *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype*. For this example, create a `HelloWorldPPA1` directory under `/var/opt/SUNWportal/portals/`*portal-ID*`/desktop/`*desktoptype* directory.

8 Copy all the template files over to the newly created directory. For example:

```
cp content.template PortalServer-DataDir/portals/
    portal-ID/desktop/desktoptype/HelloWorldPPA1
cp edit.template /var/opt/SUNWportal/portals/portal-ID
/desktop/desktoptype/HelloWorldPPA1
```

9 Create a resource bundle properties file.

The sample HelloWorldProvider includes a `HelloWorldProviderPPA1.properties` file.

a. Create a `HelloWorldProviderPPA1.properties` properties file.

b. Include the following property in the file:

```
welcome=Welcome to Hello World!!
```

c. Copy the resource bundle to the user defined class directory.

By default, the class directory to copy the resource bundle into is *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes`.

10 Specify the URL in your browser to access the HelloWorld channel inside the JSPTableContainer.

`http://`*hostname*`:`*port*`/`*portal-ID*`/dt?provider=JSPTableContainer`

▼ To Create a Provider by Extending the ProfileProviderAdapter Class

This section provides the instructions for creating a custom provider by extending the ProfileProviderAdapter. The process includes creating a sample provider that prints "Hello World!" on the provider's channel. The following sample HelloWorldProvider reads a string property from the user's display profile and allows the user to edit the string.

The sample HelloWorldProvider described here will also support `EDIT_COMPLETE`, which means this provider is responsible for generating the complete edit form with header, footer, and handling of the form submit actions. To accomplish this:

1 Create the Java class, which will generate the content for the channels backed by this provider.

For the sample HelloWorldProvider, create the class file.

```
package custom.helloworld;

import java.util.Hashtable;
import java.util.ResourceBundle;

import java.net.URL;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.sun.portal.providers.ProfileProviderAdapter;
import com.sun.portal.providers.ProviderException;
import com.sun.portal.providers.context.ProviderContext;
import com.sun.portal.providers.context.
ProviderContextException;
import com.sun.portal.providers.context.Theme;

public class HelloWorldProviderPPA2 extends
ProfileProviderAdapter {
    private ResourceBundle bundle = null;
    private final static String BANNER
        = "banner";
    private final static String BANNER_TEMPLATE
       = "banner.template";
    private final static String BORDER_COLOR
        = "borderColor";
    private final static String BORDER_WIDTH
        = "borderWidth";
    private final static String BULLET_COLOR
        = "bulletColor";
    private final static String BULLET_COLOR_JS
        = "bulletColor.js";
```

```
    private final static String BG_COLOR
        = "bgColor";
    private final static String CONTENT_LAYOUT
        = "contentLayout";
    private final static String
        CONTENT_LAYOUT_TEMPLATE =
        "contentLayout.template";
    private final static String DESKTOP_URL
        = "desktop_url";
    private final static String EDIT_PROVIDER_TEMPLATE =
        "editTemplate.template";
    private final static String ERR_MESSAGE
        = "errMessage";
    private final static String FONT_COLOR
        = "fontColor";
    private final static String FONT_FACE
        = "fontFace";
    private final static String FONT_FACE1
        = "fontFace1";
    private final static String FRONT_CONTAINER_NAME =
        "frontContainerName";
    private final static String INLINE_ERROR
        = "inlineError";
    private final static String MENUBAR
        = "menubar";
    private final static String MENUBAR_TEMPLATE
        = "menubar.template";
    private final static String MESSAGE
        = "message";
    private final static String NO_CACHE
        = "noCache";
    private final static String NO_CACHE_TEMPLATE
        = "noCache.template";
    private final static String PARENT_CONTAINER_NAME =
        "parentContainerName";
    private final static String PRODUCT_NAME
        = "productName";
    private final static String THEME_CHANNEL
        = "theme_channel";
    private final static String TITLE_BAR_COLOR
        = "titleBarColor";
    private final static String TOOLBAR_ROLLOVER
        = "toolbarRollover";
    private final static String TOOLBAR_ROLLOVER_JS =
        "toolbarRollovers.js";

/*Implement the getContent method*/
     public StringBuffer getContent(
```

```
     HttpServletRequest req, HttpServletResponse res
     ) throws ProviderException {
        Hashtable<String,Object> tags =
            getStandardTags(req);
        if (bundle == null) {
             bundle = getResourceBundle();
        } tags.put("welcome",
            bundle.getString("welcome"));
        tags.put("message",
            getStringProperty("message"));
        StringBuffer b = getTemplate
            ("content.template", tags);
        return b;
    }

/*Implement the getEdit method*/
    public StringBuffer getEdit(
    HttpServletRequest req, HttpServletResponse res)
    throws ProviderException {
        String containerName =
            req.getParameter("containerName");
        String providerName =
            req.getParameter("targetprovider");
        String editChannelName =
            req.getParameter("provider");
        // Get the standard tags,
            which will include the
            menubar, footer, etc.
        Hashtable<String,Object> tags =
            getStandardTags(req);
        tags.put("help_link",
            getHelpLink("editPage",req));
        // Provider specific tags
        tags.put(MESSAGE, getStringProperty(MESSAGE));
        tags.put("title", getTitle());
        tags.put( "providerName", providerName);
        tags.put( "contentOptions",
            getTemplate("edit.template",
            tags).toString());
        if (containerName != null) {
             tags.put(FRONT_CONTAINER_NAME, containerName);
        }
         StringBuffer b = getTemplate
            (EDIT_PROVIDER_TEMPLATE, tags);;
         return b;
    }

    /* Implement Process Edit Method */
```

```
    public URL processEdit(
    HttpServletRequest req, HttpServletResponse res
    ) throws ProviderException {
        String message = req.getParameter(MESSAGE);
        if(message != null) {
            if( !message.equals(getStringProperty(MESSAGE)) ) {
                setStringProperty(MESSAGE,message);
                return null;
            }
        } else {
             setStringProperty(MESSAGE,"");
        }
        return null;
    }

/* HelloWorld Provider supports,
        edit_type=edit_complete which means,
    the provider is responsible for generating
 a complete edit form with header,footer and html
buttons. getStandardTags method populates the
hashtable with all the standard tags */
    protected Hashtable getStandardTags
        (HttpServletRequest req) throws
        ProviderException {
        Hashtable<String,Object> tags =
            new Hashtable<String,Object>();
        ProviderContext pc = getProviderContext();
        String property = null;
        property = FONT_FACE1;
        tags.put(FONT_FACE1, getStringProperty(FONT_FACE1));
        property = PRODUCT_NAME;
        tags.put(PRODUCT_NAME,
            getStringProperty(PRODUCT_NAME));
        tags.put(PARENT_CONTAINER_NAME,
            pc.getDefaultChannelName());
        // templates
        tags.put(MENUBAR, getTemplate(MENUBAR_TEMPLATE));
        tags.put(CONTENT_LAYOUT,
            getTemplate(CONTENT_LAYOUT_TEMPLATE));
        tags.put(TOOLBAR_ROLLOVER,
            getTemplate(TOOLBAR_ROLLOVER_JS));
        tags.put(NO_CACHE, getTemplate(NO_CACHE_TEMPLATE));
        tags.put(DESKTOP_URL, pc.getDesktopURL(req));
        tags.put(BULLET_COLOR,
            getTemplate(BULLET_COLOR_JS));
         tags.put(BANNER, getTemplate(BANNER_TEMPLATE));
        // error tags (if any)
         String err = req.getParameter("error");
```

```
            if (err != null) {
                tags.put(ERR_MESSAGE, err);
                tags.put("inlineError",
                     getTemplate("inlineError.template"));
            } else {
                // no error, put in dummy
                        tags so lookup doesn't fail
                tags.put(ERR_MESSAGE, "");
                tags.put(INLINE_ERROR, "");
           }
           // theme attributes
               try {
                   tags.put( BG_COLOR, Theme.
                           getAttribute( getName(), pc,
                           BG_COLOR ));
                   tags.put( TITLE_BAR_COLOR,
                           Theme.getAttribute( getName(),
                           pc, TITLE_BAR_COLOR ));
                   tags.put( BORDER_COLOR,
                           Theme.getAttribute( getName(), pc,
                           BORDER_COLOR ));
                   tags.put( FONT_COLOR,
                           Theme.getAttribute( getName(), pc,
                           FONT_COLOR ));
                   tags.put( BORDER_WIDTH,
                           Theme.getAttribute( getName(),
                           pc, BORDER_WIDTH ));
                   tags.put( FONT_FACE,
                           Theme.getAttribute( getName(),
                           pc, FONT_FACE));
                   if( Theme.getSelectedName(getName(),
                           pc).equals
                           ( Theme.CUSTOM_THEME ) ) {
                       tags.put(THEME_CHANNEL, getStringProperty
                               ("customThemeChannel"));
                   } else {
                       tags.put(THEME_CHANNEL, getStringProperty
                               ("presetThemeChannel"));
                   }
               } catch (ProviderContextException pce) {
                   throw new ProviderException
                               ( "TemplateContainerProvider.
                           getStardardTags(): failed to
                           obtain theme related attribute ", pce );
               }
                return tags;
           }
       /* GetHelpLink for the Help button in the
```

```
        banner and footer of the edit page */
        protected String
            getHelpLink(String key, HttpServletRequest
            req) throws ProviderException{
            try {
                URL helpLink = getHelp(req, key);
                if (helpLink != null) {
                    return helpLink.toString();
                }
                else {
                    return "";
                }
            }  catch (Throwable t ) {
                throw new ProviderException
                    ("HelloWorldProvider.getHelpLink():
                        could not get help URL for " + key, t);
            }
    }
}
```

2 Compile the class file.

To compile the `HelloWorldProviderPPA2.java` file, type:

`javac -d` *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes -classpath` *PortalServer-base*`/sdk/desktop/desktopsdk.jar:`*AccessManager-base*`/lib/servlet.jar HelloWorldProviderPPA2.java`

3 Create the provider and channel definition for the display profile and load the provider and channel display profile definitions using the `psadmin add-display-profile` sub command.

The HelloWorldProvider provider display profile XML fragment (in file `HelloProviderPPA2.xml`) is shown here:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Provider name="HelloWorldProviderPPA2" class="
custom.helloworld.HelloWorldProviderPPA2">
    <Properties>
        <String name="title" value="*** TITLE ***"/>
        <String name="presetThemeChannel" value="JSPPresetThemeContainer"
        advanced="true"/>
        <String name="customThemeChannel" value="JSPCustomThemeContainer"
        advanced="true"/>
        <String name="description" value="*** DESCRIPTION ***"/>
        <String name="refreshTime" value="0"/>
        <Boolean name="isEditable" value="true"/>
        <String name="editType" value="edit_subset"/>
        <Collection name="aList">
                <String value="i'm aList"/>
```

```
            </Collection>
            <Collection name="aMap">
                <String name="title" value="Ramag"/>
                <String name="desc" value="My Map"/>
                <Boolean name="removable" value="true"/>
                <Boolean name="renamable" value="true"/>
        </Collection>
        <String name="message" value="Hello World Test Provider"/>
    </Properties>
</Provider>
```

The HelloWorldProvider channel display profile XML fragment (in file `HelloChannelPPA2.xml`) is shown here:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Channel name="HelloWorldPPA2" provider="HelloWorldProviderPPA2">
    <Properties>
        <String name="title" value="Hello World Channel"/>
        <String name="fontFace1" value="sans-serif"/>
        <String name="productName" value="Sun Java System Portal Server"/>
        <String name="editType" value="edit_complete"/>
        <String name="description">This is a test of the hello world
        provider</String>
        <String name="width" value="thin"/>
        <String name="message" value="Hello World Test!!! - non-localized "/>
        <Locale language="en" country="US">
            <String name="message" value="Hello World - I am speaking
            English in the United States!!!"/>
        </Locale>
    </Properties>
</Channel>
```

Note – See the *Sun Java System Portal Server 7.1 Command Line Reference* for more information on the `psadmin` sub commands.

---

4 Create the resource bundle properties file for the HelloWorldProvider (`HelloWorldProviderPPA2.properties`) and include the following string:

```
welcome=Welcome to Hello World!!
```

5 Copy the resource bundle properties file, `HelloWorldProviderPPA2.properties`, into *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes` directory.

6 Develop the template files for the provider.

The HelloWorldProvider requires the following template files:

`content.template`

```
<head></head>
<body bgcolor="white">
    [tag:welcome]<br><br>
    [tag:message]<br><br>
</body>
</html>
```

`edit.template` This file generates the editable fields in the form.

```
<p>
<table border="0" cellpadding="2" cellspacing="0"
width="100%">

    <tr>
        <td colspan="3" bgcolor="#333366">
            <font size=+1 face="[tag:fontFace1]"
            color="#FFFFFF"><b>Welcome</b></font>
        </td>
    </tr>

    <tr><td><br><br><br></td></tr>

    <tr>
        <td width="20%" align="RIGHT" NOWRAP>
            <font face="[tag:fontFace1]"
            color="#000000"><label
            for="message"><b>Greeting:</b>
</label></font>
        </td>
        <td width="45%">
            <font face="[tag:fontFace1]"
            size="+0">
                <input type="TEXT" name="message"
                size="50"
                maxlength="50" value="[tag:message]"
                id="message">
            </font>
        </td>
    </tr>

    <tr>
        <td width="20%"> </td>
        <td width="45%"> </td>
    </tr>

</table>
```

`editTemplate.template` This file generates a complete form

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>
    [tag:noCache]
    <TITLE>[tag:productName]</TITLE>
</HEAD>
    <BODY BGCOLOR="#FFFFFF">
        <FONT SIZE=+0 FACE="[tag:fontFace1]">
        <LINK REL="stylesheet" TYPE="text/css" HREF="
        [surl:/desktop/css/style.css]" TITLE="fonts">
        <CENTER>

        [tag:bulletColor]
        [tag:banner]

        <FORM ACTION="dt" NAME="edit_form" METHOD=POST
        ENCTYPE="application/x-www-form-urlencoded">
        <INPUT TYPE=HIDDEN NAME="action" SIZE=-1
        VALUE="process">
        <INPUT TYPE=HIDDEN NAME="provider"
        SIZE=-1 VALUE="
        [tag:providerName]">

        [tag:inlineError]

        <TABLE BORDER=0 CELLPADDING=0 CELLSPACING=3
        WIDTH="100%">
            <TR>
                <TD WIDTH="100%" VALIGN=TOP>
                    <CENTER>
                        <BR>
                        <FONT SIZE="+2"
                        FACE="[tag:fontFace1]">
                        <B>
                        Edit [tag:title]</B></FONT>
                        <BR>
                        [tag:contentOptions]
                    </CENTER>
                </TD>
            </TR>
        </TABLE>

        <BR>

        <FONT SIZE=+0 FACE="[tag:fontFace1]">
```

```
                <INPUT TYPE=SUBMIT NAME="Submit"
                VALUE="Finished"
                CLASS="button">
                <INPUT TYPE=BUTTON OnClick="
                location='[tag:desktop_url]'
                "VALUE="Cancel" CLASS="button">
                </font>

                <br>

                <P>
                </FORM>
                <BR>
                [tag:menubar]
                </font>
            </BODY>
        </HTML>
```

7 Create the channel directory under the template root directory and copy the templates for the provider into the channel directory under the templates root directory.

By default, the template root directory is *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype*. For this example, create a `HelloWorldPPA2` directory under *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype* directory. To copy the templates, for example, type:

```
cp content.template PortalServer-DataDir/portals/
portal-ID/desktop/desktoptype/HelloWorldPPA2
cp edit.template PortalServer-DataDir/portals/
portal-ID/desktop/desktoptype/HelloWorldPPA2
cp editTemplate.template PortalServer-DataDir
/portals/portal-ID/desktop/desktoptype
/HelloWorldPPA2
```

8 Specify the URL to access the channel from a browser.

`http://`*hostname*`:`*port*`/`*portal-ID*`/dt?provider=HelloWorldPPA2`

To display the provider's channel as one of the leaf channels for a container, you have to add the channel into the container's available and selected list. To accomplish this:

9 Add it to the Available and Selected list for the container.

For example, to add the HelloWorldProvider to the TemplateTableContainer, use the `psadmin modify-display-profile` sub command. For more information on the `psadmin` utility and its sub commands, see the *Sun Java System Portal Server 7.1 Command Line Reference*.

10 Login to the Desktop and specify the URL in your browser to access the HelloWorld Channel inside the TemplateTableContainer.

`http://`*hostname*`:`*port*`/`*portal-ID*`/dt?provider=TemplateTableContainer`

Utilize the `psadmin` utility for transporting channels and providers. Using the `psadmin` command, the contents of the `helloworld.par` can be imported to as HelloWorld channel in a different Portal Server software installation. For more information on the `psadmin` utility and its sub commands, see the *Sun Java System Portal Server 7.1 Command Line Reference*.

Extending the PropertiesFilter Class

Developers who wish to implement an arbitrary filter to selectively look up properties with certain criteria and/or to store properties that are specific to certain setting can extend the PropertiesFilter class.

▼ To create a custom filter by extending the PropertiesFilter class

1 Create the Java class, which will implement the filter.

The class file for the sample DateLaterThanPropertiesFilter is shown here:

```
package com.acme.filters;

import java.util.Iterator;
import java.util.Date;
import java.text.DateFormat;
import java.text.ParseException;

import com.sun.portal.providers.context.PropertiesFilter;
import com.sun.portal.providers.context.PropertiesFilterException;
import com.sun.portal.providers.context.ProviderContext;
public class DateLaterThanPropertiesFilter extends PropertiesFilter {

    private static final DateFormat dateFormat =
        DateFormat.getDateInstance(DateFormat.SHORT);
    private Date date = null;

    public DateLaterThanPropertiesFilter() {
        super();
    }

    protected void init(String value, boolean required)
        throws PropertiesFilterException {
```

```
        super.init(value, required);
        try {
            date = dateFormat.parse(value);
        } catch (ParseException pe) {
        throw new PropertiesFilterException("DateLaterThanPropertiesFilter:
             ", pe);
        }
    }

    public String getCondition() {
        return "dateLaterThan";
    }

    public boolean match(String condition, String value) throws
        PropertiesFilterException {
        if (!condition.equals("dateLaterThan")) {
            return false;
        }
        Date cdate = null;
        try {
            cdate = dateFormat.parse(value);
        } catch (ParseException pe) {
            throw new PropertiesFilterException
                ("DateLaterThanPropertiesFilter.match(): ", pe);
        }
        return cdate.after(date);
    }
}
```

2 Compile the class file.

To compile the `DateLaterThanPropertiesFilter.java` file, type:

```
javac -d /var/opt/SUNWportal/portals/portal-ID
/desktop/classes -classpath PortalServer-base/sdk/desktop/desktopsdk.jar:
AccessManager-base/lib/servlet.jar DateLaterThanPropertiesFilter.java
```

3 Create the provider display profile XML fragment for the conditional properties.

The sample display profile XML fragment for the DateLaterThanPropertiesFilter is here:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">
<Properties>
    <String name="foo" value="bar">
    <ConditionalProperties condition="locale" value="en">
        <String name="foo" value="bar"/>
        <ConditionalProperties condition="dateLaterThan" value="03/03/2003">
            <ConditionalProperties condition="client" value="nokia">
                <String name="foo" value="en 03/03/2003 nokia">
```

```
            </ConditionalProperties>
        </ConditionalProperties>
            <ConditionalProperties condition="client" value="nokia">
                <String name="foo" value="en nokia">
        </ConditionalProperties>
    </ConditionalProperties>
</Properties>
```

4 Load the display profile using the `psadmin modify-display-profile` sub command.

See the *Sun Java System Portal Server 7.1 Command Line Reference* for more information on this sub command. Alternatively, you can also use the administration console to add and/or modify display profile. See the Portal Server administration console online help for more information.

5 Implement the code to access the filter in your custom provider class file where you wish to implement this filter.

For example, in your provider, you can access the sample DateLaterThanPropertiesFilter by doing one of the following:

- The following Provider implementation for `DateLaterThanPropertiesFilter` will return "en 03/03/2003 nokia"

```
List pflist = new List();
pflist.add(getProviderContext().getLocalePropertiesFilter( "en", true));
String filter = "com.acme.filters.DateLaterThanPropertiesFilter" ;
pflist.add(getProviderContext().getPropertiesFilter( filter, "02/02/2003", true));
pflist.add(getProviderContext().getClientPropertiesFilter( "nokia", true));
getStringProperty(getName(), "foo", pflist);
```

- The following Provider implementation for `DateLaterThanPropertiesFilter` will return "en nokia"

```
List pflist = new List();
pflist.add(getProviderContext().getLocalePropertiesFilter( "en", true));
String filter = "com.acme.filters.DateLaterThanPropertiesFilter" ;
pflist.add(getProviderContext().getPropertiesFilter
( filter, "04/04/2003", false)); // this filter is optional
pflist.add(getProviderContext().getClientPropertiesFilter( "nokia", true));
getStringProperty(getName(), "foo", pflist);
```

◆ ◆ ◆ CHAPTER 8

Extending the Leaf Providers

This chapter includes detailed instructions for:

Extending the JSPProvider

This sections contains some sample extensions to the JSPProvider.

Example 1

In the following example, the JSPProvider `getContent()` method is overridden to make a connection to the database to get the address book from the database and use a JSP to display the content of the address book for each user.

▼ To Extend the JSPProvider

1 Extend the JSPProvider and develop the AddressBookProvider class.

The AddressBookProvider class overrides the JSPProvider's `getContent()` method which reads the user's address book. The `getContent()` method gets the address book for the user who is logged in to the Desktop.

`AddressBookProvider.java` File

```
package custom;

import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletRequest;
```

```
import com.sun.portal.providers.ProviderException;
import com.sun.portal.providers.jsp.JSPProvider;

public class AddressBookProvider extends JSPProvider {
    public StringBuffer getContent (
    HttpServletRequest req, HttpServletResponse res
    ) throws ProviderException {
        AddressBook addrBook = new custom.AddressBook
        (getProviderContext().getUserID());
        req.setAttribute("addressBook", addrBook);
        return super.getContent(req, res);
    }
}
```

2 Develop the AddressBook class.

The class AddressBook is responsible for making a connection to the database and getting the user's address book from the database.

3 Compile both classes and put them in the provider class base directory.

The default directory for the class file is *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes`. That is, build a JAR and copy the JAR into the provider class base directory. Or, just copy the class files into the provider class base directory.

4 Develop the display profile XML fragment for this provider and this provider's channel.

The display profile fragment for the AddressBookProvider's provider is saved in `AddressBookProvider.xml` file and the display profile fragment for the AddressBookProvider's channel is saved in `AddressBookChannel.xml` file.

`AddressBookProvider.xml` File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Provider name="AddressBookProvider" class="custom.AddressBookProvider">
    <Properties>
        <String name="title" value="Address Book Provider"/>
        <String name="description" value="My Addressbook"/>
        <String name="refreshTime" value="0" advanced="true"/>
        <Boolean name="isEditable" value="false" advanced="true"/>
        <String name="editType" value="edit_subset" advanced="true"/>
        <String name="width" value="thin" advanced="true"/>
        <String name="helpURL" value="desktop/usedesk.htm" advanced="true"/>
        <String name="fontFace1" value="Sans-serif"/>
        <String name="productName" value="Sun Java System Portal Server"/>
        <String name="contentPage" value="addressBook.jsp"/>
    </Properties>
</Provider>
```

In `AddressBookChannel.xml`, note that the AddressBookProvider channel definition is embedded in an existing table container, JSPTableContainer.

`AddressBookChannel.xml` File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Container name="JSPTableContainer" provider="JSPTableContainerProvider">
    <Properties>
        <Collection name="Personal Channels">
            <String name="AddressBook"/>
        </Collection>
        <Collection name="channelsColumn">
            <String name="AddressBook" value="3"/>
        </Collection>
    </Properties>
    <Available>
        <Reference value="AddressBook"/>
    </Available>
    <Selected>
        <Reference value="AddressBook"/>
    </Selected>
    <Channels>
        <Channel name="AddressBook" provider="AddressBookProvider">
            <Properties>
            </Properties>
        </Channel>
    </Channels>
</Container>
```

5 Use the `psadmin` command to upload the display profile fragments for this provider.

For the AddressBookProvider, use the `psadmin add-display-profile` command to upload the `AddressBookProvider.xml` file and use the `psadmin modify-display-profile` command to upload the `AddressBookChannel.xml` file XML fragments in the display profile. For more information on these sub commands, see the *Sun Java System Portal Server 7.1 Command Line Reference*.

6 Develop the JSP file for the provider.

For the sample AddressBookProvider, the JSP in `addressBook.jsp` can be used to display the content of the address book for each user. However, the details of displaying the address book are not shown in `addressBook.jsp`.

`addressBook.jsp` File

```
<%-- addressBook.jsp --%>
<%@page import="custom.AddressBookProvider" %>
<%@page import="custom.AddressBook" %>
```

```
<%
AddressBookProvider addressBookProvider = (AddressBookProvider)
pageContext.getAttribute("JSPProvider");
AddressBook addrBook = (AddressBook)pageContext.getAttribute
("addressBook",PageContext.REQUEST_SCOPE);
%>

<%-- Display the address book --%>
```

7 Create a channel directory under the template root directory.

By default, the template root directory is *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype*. For this example, create an AddressBook directory under *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype* directory.

8 Copy all the JSP files over to the newly created directory (in "Example 1" on page 115). For example, type:

`cp addressBook.jsp` *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype*`/AddressBook`

9 Login to the Desktop and specify the URL in your browser to access the AddressBook channel inside the JSPTableContainer.

`http://`*hostname*`:`*port*`/`*portal-ID*`/dt?provider=JSPTableContainer`

Example 2

In the following example, the SimpleUserInfoProvider overrides the `processEdit()` method of the JSPProvider. It reads the common name that is input in the edit page JSP, and then it is read from the `processEdit()` method. The value is set to the backend store. To accomplish this:

▼ To Develop the SimpleUserInfoProvider

1 Extend the JSPProvider and develop the SimpleUserInfoProvider class.

The SimpleUserInfoProvider class overrides the JSPProvider `processEdit()` method as shown in "Example 2" on page 118.

```
SimpleUserInfoProvider.java File
package custom;

import java.net.URL;

import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletRequest;
```

```
import com.sun.portal.providers.ProviderException;
import com.sun.portal.providers.jsp.JSPProvider;

public class SimpleUserInfoProvider extends JSPProvider {
    public URL processEdit(
    HttpServletRequest req, HttpServletResponse res
    ) throws ProviderException {
        //get the common name from the request and set it into the backend store
        getProviderContext().setStringAttribute("cn",(String)req.getParameter("cn"));
        return null;
    }
}
```

2 Compile the class and put it in the provider class base directory.

The default directory for the class file is *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes`. That is, build the JAR and copy the JAR into the provider class base directory. Or, just copy the class as a file into the provider class base directory. To compile the `SimpleUserInfoProvider.java` file, type:

`javac -d` *PortalServer-DataDir*`/portals/`
portal-ID`/desktop/classes -classpath` *PortalServer-base*`/sdk/desktop/desktopsdk.jar:`
AccessManager-base`/lib/servlet.jar SimpleUserInfoProvider.java`

3 Develop the display profile XML fragment for this provider and this provider's channel.

The display profile fragment for the SimpleUserInfoProvider's provider is saved in `SimpleUIProvider.xml` file and the display profile fragment for the SimpleUserInfoProvider's channel is saved in `SimpleUIPChannel.xml` file.

```
SimpleUIProvider.xml File
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Provider name="SimpleUserInfoProvider" class="custom.SimpleUserInfoProvider">
    <Properties>
        <String name="title" value="Simple User Information Provider"/>
        <String name="description" value="My UserInformation"/>
        <String name="refreshTime" value="0" advanced="true"/>
        <Boolean name="isEditable" value="false" advanced="true"/>
        <String name="editType" value="edit_subset" advanced="true"/>
        <String name="width" value="thin" advanced="true"/>
        <String name="helpURL" value="desktop/usedesk.htm" advanced="true"/>
        <String name="fontFace1" value="Sans-serif"/>
        <String name="productName" value="Sun Java System Portal Server"/>
        <String name="contentPage" value="simpleUserInfoContent.jsp"/>
        <String name="editPage" value="simpleUserInfoEdit.jsp"/>
    </Properties>
</Provider>
```

In `SimpleUIPChannel.xml`, note that the SimpleUserInfoProvider channel definition is embedded inside JSPTableContainer thus making the channel only visible via the container.

```
SimpleUIPChannel.xml File
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Container name="JSPTableContainer" provider="JSPTableContainerProvider">
    <Properties>
        <Collection name="Personal Channels">
            <String name="SimpleUserInfo"/>
        </Collection>
        <Collection name="channelsColumn">
            <String name="SimpleUserInfo" value="3"/>
        </Collection>
    </Properties>
    <Available>
        <Reference value="SimpleUserInfo"/>
    </Available>
    <Selected>
        <Reference value="SimpleUserInfo"/>
    </Selected>
    <Channels>
        <Channel name="SimpleUserInfo" provider="SimpleUserInfoProvider">
            <Properties>
                <String name="processPage" value="simpleUserInfoDoedit.jsp"/>
            </Properties>
        </Channel>
    </Channels>
</Container>
```

4 Use the `psadmin` command to upload the display profile fragments for this provider.

For the SimpleUserInfoProvider, use the `psadmin add-display-profile` command to upload the `SimpleUIProvider.xml` file and use the `psadmin modify-display-profile` command to upload the `SimpleUIPChannel.xml` file XML fragments in the display profile. For more information on these sub commands, see the *Sun Java System Portal Server 7.1 Command Line Reference*.

5 Create a channel directory under the template root directory.

By default, the template root directory is *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype*. For this example, create an `SimpleUserInfo` directory under *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype* directory.

6 **Develop and copy all the JSP files over to the newly created directory (in Step 5).**

For example, develop the `simpleUserInfoContent.jsp` and `simpleUserInfoEdit.jsp` files and copy them over to the newly created *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype*`/SimpleUserInfo` directory.

7 **Login to the Desktop and specify the URL in your browser to access the SimpleUserInfo channel inside the JSPTableContainer.**

`http://`*hostname*`:`*port*`/`*portal-ID*`/dt?provider=JSPTableContainer`

Extending the URLScraperProvider

The following sample extends the URLScraperProvider and overrides the `getURL()` method to get a random URL from a pool of URLs defined in the display profile. This sample can be used to display a random image from a pool of images; that is, the following `getURL()` method selects an URL at random from a pool of URLs where each URL is pointing to a HTML file containing the image.

▼ To Extend the URLScraperProvider

1 **Extend the URLScraperProvider and develop the custom class file.**

The CustomURLScraperProvider class file overrides the URLScraperProvider `getURL()` method as shown in "Extending the URLScraperProvider" on page 121. The CustomURLScraperProvider gets the URL property for the provider. This is the URL from where the contents are fetched.

```
CustomURLScraperProvider.java File
package custom;

import java.util.List;
import java.util.Hashtable;
import java.util.Random;

import com.sun.portal.providers.urlscraper.URLScraperProvider;
import com.sun.portal.providers.ProviderException;
import com.sun.portal.providers.context.ProviderContextException;

public class CustomURLScraperProvider
extends URLScraperProvider {
    protected String getURL() throws ProviderException {
        List urlList = getListProperty("urlPool");
        int size = urlList.size();
        if ( size == 1) {
            return (String)urlList.get(0);
```

```
        }
        Random rand = new Random();
        int r = rand.nextInt(size-1);
        return (String)urlList.get(r);
    }
}
```

2 Compile the class and put it in the provider class base directory.

The default directory for the class file is *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes`. That is, build the JAR and copy the JAR into the provider class base directory. Or, just copy the class as a file into the provider class base directory. To compile the `CustomURLScraperProvider.java` file, type:

```
javac -d /var/opt/SUNWportal/portals/portal-ID/desktop/classes -classpath
PortalServer-base/sdk/desktop/desktopsdk.jar:AccessManager-base
/lib/servlet.jar CustomURLScraperProvider.java
```

3 Develop the display profile XML fragment for this provider and this provider's channel.

The display profile fragment for the CustomURLScraperProvider's provider is saved in `CustomURLSProvider.xml` file and the display profile fragment for the CustomURLScraperProvider's channel is saved in `CustomURLSChannel.xml` file.

```
CustomURLSProvider.xml File
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Provider name="CustomURLScraperProvider" class="custom.CustomURLScraperProvider">
    <Properties>
        <String name="title" value=" Custom UrlScraper Channel"/>
        <String name="description" value="This channel displays the urls at random."/>
        <Boolean name="isEditable" value="false" advanced="true"/>
        <String name="urlScraperRulesetID" value="default_ruleset"/>
        <String name="width" value="thick"/>
        <String name="refreshTime" value="0" advanced="true"/>
        <Collection name="urlPool">
            <String value="http://
                localhost:
                port/
                file1.
                html"/>
            <String value="http://
                localhost:
                port/
                file2.html"/>
        </Collection>
        <String name="fontFace1" value="Sans-serif"/>
        <String name="productName" value="Sun Java System Portal Server"/>
        <Boolean name="cookiesToForwardAll" value="true"/>
```

```
        <String name="inputEncoding" value="UTF-8"/>
        <Collection name="cookiesToForwardList">
        </Collection>
        <Integer name="timeout" value="100"/>
    </Properties>
</Provider>
```

`CustomURLSChannel.xml` File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Channel name="CustomURLScraper" provider="CustomURLScraperProvider">
    <Properties>
        <String name="refreshTime" value="600" advanced="true"/>
        <Collection name="urlPool">
            <String value="http://
                localhost:
                port/
                file1.html"/>
            <String value="http://
                localhost:
                port/
                file2.html"/>
        </Collection>
    </Properties>
</Channel>
```

4 Use the `psadmin` command to upload the display profile fragments for this provider.

For the CustomURLScraperProvider, use the `psadmin add-display-profile` command to upload the `CustomURLSProvider.xml` and use the `psadmin modify-display-profile` command to upload the `CustomURLSChannel.xml` file XML fragments in the display profile. For more information on these commands, see the *Sun Java System Portal Server 7.1 Command Line Reference*.

5 Include the new channel in one of the existing containers.

The CustomURLScraperProvider channel will be displayed in JSPTableContainer. To add the channel in the JSPTableContainer from the administration console, follow instructions in Portal Server administration console online help. To add the channel in the JSPTableContainer manually:

a. Add the CustomURLScraperProvider channel XML fragment (in the `CustomURLSContainer.xml` file) for the JSPTableContainerProvider.

The CustomURLScraperProvider container XML fragment is shown below.

`CustomURLSContainer.xml` File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
```

```
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Container name="JSPTableContainer" provider="JSPTableContainerProvider">
    <Properties>
        <Collection name="Personal Channels">
            <String value="CustomURLScraper"/>
        </Collection>
        <Collection name="channelsRow">
            <String name="CustomURLScraper" value="3"/>
        </Collection>
    </Properties>
    <Available>
            <Reference value="CustomURLSscraper"/>
    </Available>
    <Selected>
        <Reference value="CustomURLScraper"/>
    </Selected>
    <Channels>
    </Channels>
</Container>
```

b. Use the `psadmin modify-display-profile` sub command to add the channel to a container.

If you do not specify a parent object with the `-p` option, the channel is added at the root level.

6 Change directories to *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes` and copy the properties file for the provider into this directory.

For example, for the CustomURLScraperProvider:

a. Change directories to *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes`.

b. Type `cp URLScraperProvider.properties` *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes/CustomURLScraperProvider.properties`.

7 Login to the Desktop and specify the URL in your browser to access the CustomURLScraperProvider channel inside the JSPTableContainer.

`http://`*hostname*`:`*port*`/`*portal-ID*`/dt?provider=JSPTableContainer`

Extending the XMLProvider

In the following example, the XMLProvider `getXML()` method is overridden to get the content out of a database. In the CustomXMLProvider, the XML content is retrieved from a database.

▼ To Extend the XMLProvider

1 Develop the CustomXMLProvider class file and override the `getXML()` method in the CustomXMLProvider's class file.

The CustomXMLProvider class file (see below) extends the XMLProvider and includes only the overriding `getXML()` method.

```
CustomXMLProvider.java File
package custom;

import com.sun.portal.providers.ProviderException;
import com.sun.portal.providers.context.ProviderContext;
import com.sun.portal.providers.xml.XMLProvider;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class CustomXMLProvider extends XMLProvider {
    protected StringBuffer getXML(
    HttpServletRequest req,HttpServletResponse res
    ) throws ProviderException {
        String xmlURL = getURL();
        StringBuffer xmlbuf = new StringBuffer();

        // It must be http, https or file i.e http://<url>, https://<url>
        // or file:////<path>
        if (xmlURL != null && xmlURL.length() != 0) {
            String proto = xmlURL.substring(0,xmlURL.indexOf(':'));
            if((proto.equalsIgnoreCase("http")) || (proto.equalsIgnoreCase
                ("https")) || (proto.equalsIgnoreCase("file"))) {
                try {
                    xmlbuf = super.getXML(req, res);
                } catch(ProviderException pr) {
                    getProviderContext().debugError("XMLProvider:getXML(): ",pr);
                }
                if (xmlbuf != null) {
                    return xmlbuf;
                } else {
                    throw new ProviderException("XMLProvider.getXML():
                            Getting XML failed from source");
                }
```

```
            }
        } else {
            // Open database connection
            // Read the XML contents into a stringbuffer (xmlbuf)
            // Close the database connection
        }
        return xmlbuf;
    }
}
```

2 Compile the class and put it in the provider class base directory.

That is, build the JAR and copy the JAR into the provider class base directory. Or, just copy the class as a file into the provider class base directory. The default directory for the class file is *PortalServer-DataDir*/portals/*portal-ID*/desktop/classes. To compile the CustomXMLProvider.java file, type:

javac -d *PortalServer-DataDir*/portals/*portal-ID*/desktop/classes -classpath *PortalServer-base*/sdk/desktop/desktopsdk.jar:*AccessManager-base*/lib/servlet.jar CustomXMLProvider.java

3 Develop the display profile XML fragment for this provider and this provider's channel.

The display profile fragment for the CustomXMLProvider's provider is saved in CustomXMLProvider.xml file and the display profile fragment for the CustomXMLProvider's channel is saved in CustomXMLChannel.xml file.

CustomXMLProvider.xml File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Provider name="CustomXMLProvider" class="custom.CustomXMLProvider">
    <Properties>
        <String name="title" value="*** XML Provider ***"/>
        <String name="description" value="*** DESCRIPTION ***"/>
        <String name="width" value="thick"/>
        <String name="refreshTime" value="0" advanced="true"/>
        <Boolean name="isEditable" value="false" advanced="true"/>
        <String name="helpURL" value="desktop/xmlchann.htm" advanced="true"/>
        <String name="fontFace1" value="Sans-serif"/>
        <String name="productName" value="Sun Java System Portal Server"/>
        <String name="url" value=""/>
        <String name="xslFileName" value="html_stockquote.xsl"/>
        <Integer name="timeout" value="100"/>
        <String name="inputEncoding" value="iso-8859-1"/>
        <String name="urlScraperRulesetID" value="default_ruleset"/>
        <Boolean name="cookiesToForwardAll" value="true"/>
        <Collection name="cookiesToForwardList">
        </Collection>
```

```
    </Properties>
</Provider>
```

`CustomXMLChannel.xml` File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Channel name="CustomXML" provider="CustomXMLProvider">
    <Properties>
        <String name="refreshTime" value="600" advanced="true"/>
        <String name="title" value="XML Test Channel"/>
        <String name="description" value="This is a test of the CustomXMLProvider"/>
        <String name="url" value=""/>
        <String name="xslFileName" value="html_stockquote.xsl"/>
    </Properties>
</Channel>
```

4 Use the `psadmin` command to upload the display profile fragments for this provider.

For the CustomXMLProvider, use the `psadmin add-display-profile` sub command to upload the `CustomXMLProvider.xml` file and the `CustomXMLChannel.xml` file XML fragments in the display profile. For more information on this sub command, see the *Sun Java System Portal Server 7.1 Command Line Reference*.

5 Include the new channel in one of the existing containers.

The CustomXMLProvider channel will be displayed in JSPTableContainer. To add the channel in the JSPTableContainer from the administration console, follow instructions in the Portal Server administration console online help. To add the channel in the JSPTableContainer manually:

a. Develop the CustomXMLProvider channel XML fragment (in the `CustomXMLContainer.xml` file) for the JSPTableContainer.

The CustomXMLProvider container fragment is shown below.

`CustomXMLContainer.xml File`

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Container name="JSPTableContainer" provider="JSPTableContainerProvider">
    <Properties>
        <Collection name="Personal Channels">
            <String value="CustomXML"/>
        </Collection>
        <Collection name="channelsRow">
            <String name="CustomXML" value="3"/>
        </Collection>
    </Properties>
```

```
    <Available>
            <Reference value="CustomXML"/>
    </Available>
    <Selected>
            <Reference value="CustomXML"/>
    </Selected>
    <Channels>
    </Channels>
</Container>
```

b. Use the `psadmin modify-display-profile` sub command to add the channel to a container.

If you do not specify a parent object with the `-p` option, the channel is added at the root level.

6 Change directories to *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes` and copy the properties file for the provider into this directory.

For example, for the CustomXMLProvider:

a. Change directories to *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes`.

b. Type `cp XMLProvider.properties` *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes/CustomXMLProvider.properties`.

7 Login to the Desktop and specify the URL in your browser to access the CustomXMLProvider channel inside the JSPTableContainer.

`http://`*hostname*`:`*port*`/`*portal-ID*`/dt?provider=JSPTableContainer`

◆ ◆ ◆ CHAPTER 9

Extending the Container Providers

This chapter contains instructions for:

Extending the JSPSingleContainerProvider

In the following example, the CustomSingleContainerProvider `getSelectedChannel(HttpServletRequest req)` method overrides the `JSPSingleContainerProvider.getSelectedChannel()` method to return the first channel from the list whose width is `full_top`.

▼ To Develop the CustomSingleContainerProvider

1 Extend the JSPSingleContainerProvider and develop the CustomSingleContainerProvider class file.

The CustomSingleContainerProvider class overrides the `JSPSingleContainerProvider.getSelectedChannel()` method as shown below.

```
package custom;

import com.sun.portal.providers.containers.jsp.single.
JSPSingleContainerProvider;
import com.sun.portal.providers.ProviderException;
import com.sun.portal.providers.Provider;
import com.sun.portal.providers.ProviderWidths;
import com.sun.portal.providers.context.ContainerProviderContext;

import javax.servlet.http.HttpServletRequest;
```

```
import java.util.List;

public class CustomSingleContainerProvider extends
JSPSingleContainerProvider {
    public String getSelectedChannel(HttpServletRequest req)
        throws ProviderException {
            List selectedChannels = getSelectedChannels();
        for (int i=0; i<selectedChannels.size(); i++) {
              String channel = (String)selectedChannels.get(i);
            Provider p = getContainerProviderContext().getProvider
                    (req, getName(), channel);
            if (p.getWidth() == ProviderWidths.WIDTH_FULL_TOP) {
               return channel;
            }
        }
        return (String)selectedChannels.get(0);
    }
}
```

2 Compile the class and put it in the provider class base directory.

The default directory for the class file is *PortalServer-DataDir*/portals/*portal-ID*/desktop/classes. That is, build the JAR and copy the JAR into the provider class base directory. Or, just copy the class as a file into the provider class base directory. To compile the CustomSingleContainerProvider.java file, enter:

```
javac -d PortalServer-DataDir/portals/
portal-ID/desktop/classes -classpath PortalServer-base/sdk/desktop/desktopsdk.jar:
AccessManager-base/lib/servlet.jar CustomSingleContainerProvider.java
```

3 Develop the display profile XML fragments for this provider.

The display profile fragment for the CustomSingleContainerProvider's provider is saved in CustomSCProvider.xml file (see below) and the display profile fragment for the CustomSingleContainerProvider's channel is saved in CustomSCChannel.xml file (see below).

CustomSCProvider.xml file

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Provider name="CustomSingleContainerProvider" class="
custom.CustomSingleContainerProvider">
    <Properties>
        <String name="contentPage" value="single.jsp"/>
        <Boolean name="showExceptions" value="false"/>
        <String name="title" value=" Single Container Provider "/>
        <String name="description" value=" DESCRIPTION "/>
        <String name="refreshTime" value="0"/>
        <String name="width" value="thin"/>
```

```
        <String name="fontFace1" value="Sans-serif"/>
        <String name="productName" value="Sun Java System Portal Server"/>
        <Boolean name="isEditable" value="true"/>
        <String name="editType" value=""/>
        <String name="editContainerName" value="JSPEditContainer"/>
        <String name="presetThemeChannel" value="JSPPresetThemeChannel"/>
        <String name="customThemeChannel" value="JSPCustomThemeChannel"/>
    </Properties>
</Provider>
```

`CustomSCChannel.xml` File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Container name="CustomSingleContainer"
provider="CustomSingleContainerProvider" advanced="false">
    <Properties>
        <String name="title" value="JSP custom single container Channel"/>
        <String name="description" value="This is a test for single
        containers"/>
        <String name="contentPage" value="single.jsp"/>
    </Properties>
    <Available>
        <Reference value="Search"/>
        <Reference value="Bookmark"/>
    </Available>
    <Selected>
        <Reference value="Search"/>
    </Selected>
    <Channels>
    </Channels>
</Container>
```

4 Use the `psadmin` command to upload the display profile fragments for this provider.

For the CustomSingleContainerProvider, use the `psadmin add-display-profile` sub command to upload the `CustomSCProvider.xml` file and `CustomSCChannel.xml` file XML fragments into the display profile. See the *Sun Java System Portal Server 7.1 Command Line Reference* for more information on this sub command.

5 Create a new directory for the provider in *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype* directory. The directory for the provider is typically named after the provider's channel.

For the sample CustomSingleContainerProvider, create a directory called `CustomSingleContainer` in *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype* directory.

6 Develop and copy the JSP files for the provider in the newly created directory.

For the CustomSingleContainerProvider, copy files from *PortalServer-DataDir*/portals/*portal-ID*/desktop/default/JSPSingleContainerProvider to the *PortalServer-DataDir*/portals/*portal-ID*/desktop/*desktoptype*/CustomSingleContainer directory.

a. Modify single.jsp **file in the** *PortalServer-DataDir*/portals/*portal-ID*/desktop/*desktoptype*/CustomSingleContainer **directory as shown here:**

```
<%-- Copyright 2001 Sun Microsystems, Inc. All rights reserved.
PROPRIETARY/CONFIDENTIAL. Use of this product is subject to license terms. --%>
<%-- single.jsp --%>
<%@ taglib uri="/tld/desktop.tld" prefix="dt" %>
<%@ taglib uri="/tld/desktopSingle.tld" prefix="dtsingle" %>
<%@ page import="custom.CustomSingleContainerProvider" %>
<%@ page session="false" %>
<jsp:include page="header.jsp" flush="true"/>
<dt:obtainContainer container="$JSPProvider">
<dtsingle:singleContainerProvider>
<dtsingle:obtainSelectedChannel>
<img src="<dt:scontent/>/desktop/images/nothing.gif" height="8" width="0"
border="0" alt=""><br>
<% CustomSingleContainerProvider csp =
(CustomSingleContainerProvider)pageContext.getAttribute("JSPProvider"); %>
<img src="<dt:scontent/>/desktop/images/nothing.gif" height="8" width="0"
border="0" alt=""><br>
<% String selected = csp.getSelectedChannel(request);
out.println(csp.getContainerProviderContext().getContent(request, response,
null, selected)); %>
</dtsingle:obtainSelectedChannel>
</dtsingle:singleContainerProvider>
</dt:obtainContainer>
<jsp:include page="menubar.jsp" flush="true"/>
<%@ include file="footer.html" %>
```

b. Add a FULL_TOP **channel to the container provider before testing the container provider.**

For example, for the CustomSingleContainerProvider, add the Search channel via the Portal Server administration console.

7 Login to the Desktop and specify the URL in your browser to access the CustomSingleContainerProvider.

http://*hostname*:*port*/*portal-ID*/dt?provider=CustomSingleContainer

Extending the JSPTableContainerProvider

This section includes some sample providers that extend JSPTableContainerProvider and override methods in the JSPTableContainerProvider API.

Example 1: Developing CustomTableContainerProvider

The following is an example of a container provider that selects channels dynamically based on the time of the day.

▼ To Develop the CustomTableContainerProvider

1 Extend JSPTableContainerProvider and develop the provider class file.

For CustomTableContainerProvider, the CustomTableContainerProvider class is implemented by extending the JSPTableContainerProvider class as shown in "Example 1: Developing CustomTableContainerProvider" on page 133.

```
package custom;

import com.sun.portal.providers.containers.jsp.table.JSPTableContainerProvider;
import com.sun.portal.providers.ProviderException;

import java.util.List;
import java.util.ArrayList;
import java.util.Calendar;

public class CustomTableContainerProvider extends JSPTableContainerProvider {
    private Calendar rightNow = Calendar.getInstance();
        public List getSelectedChannels() throws ProviderException {
        List selectedChannels = super.getSelectedChannels();
        List<String> newList = new ArrayList<String>();
        if (rightNow.AM_PM == rightNow.AM) {
            List amList = (List)getListProperty("amList");
            for (int i = 0; i < amList.size(); i++) {
                String channel = (String)amList.get(i);
                if (selectedChannels.contains(channel)) {
                    newList.add(channel);
                }
            }
            return newList;
        } else {
            List pmList = (List)getListProperty("pmList");
            for (int i = 0; i < pmList.size(); i++) {
                String channel = (String)pmList.get(i);
```

```
                if (selectedChannels.contains(channel)) {
                    newList.add(channel);
                }
            }
            return newList;
        }
    }
}
```

2 Compile the class and put it in the provider class base directory.

The default directory for the class file is *PortalServer-DataDir*/portals/*portal-ID*/desktop/classes. That is, build the JAR and copy the JAR into the provider class base directory. Or, just copy the class as a file into the provider class base directory. To compile the CustomTableContainerProvider.java file, enter:

```
javac -d PortalServer-DataDir/portals/portal-ID/
desktop/classes -classpath PortalServer-base/sdk/desktop/desktopsdk.jar:
AccessManager-base/lib/servlet.jar CustomTableContainerProvider.java
```

3 Develop the display profile XML fragments for this provider and this provider's channel.

The display profile fragment for the CustomTableContainerProvider's provider is saved in CustomTCProvider.xml file and the display profile fragment for the CustomTableContainerProvider's channel is saved in CustomTCChannel.xml file.

CustomTCProvider.xml File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Provider name="CustomTableContainerProvider" class="
custom.CustomTableContainerProvider">
    <Properties>
        <String name="contentPage" value="toptable.jsp"/>
        <Integer name="timeout" value ="1800"/>
        <Integer name="layout" value="1"/>
        <Boolean name="showExceptions" value="false"/>
        <Boolean name="parallelChannelsInit" value="false"/>
        <String name="title" value="Table Container Provider"/>
        <String name="description" value="DESCRIPTION"/>
        <String name="refreshTime" value="0"/>
        <String name="width" value="thin"/>
        <String name="fontFace1" value="Sans-serif"/>
        <String name="Desktop-fontFace1" value="Sans-serif"/>
        <String name="productName" value="Sun Java System Portal Server"/>
        <String name="presetThemeChannel" value="JSPPresetThemeContainer"/>
        <String name="customThemeChannel" value="JSPCustomThemeContainer"/>
        <Boolean name="refreshParentContainerOnly" value="false" advanced="true"/>
        <Boolean name="isEditable" value="true"/>
        <String name="editType" value="edit_complete"/>
```

```
<String name="editContainerName" value="JSPEditContainer"/>
<Integer name="thin_popup_height" value="200"/>
<Integer name="thin_popup_width" value="500"/>
<Integer name="thick_popup_height" value="300"/>
<Integer name="thick_popup_width" value="600"/>
<Integer name="fullwidth_popup_height" value="500"/>
<Integer name="fullwidth_popup_width" value="600"/>
<Collection name="categories">
    <String value="Personal Channels"/>
    <String value="Sample Channels"/>
    <String value="News Channels"/>
    <String value="Search Channels"/>
</Collection>
<Collection name="Personal Channels">
    <String value="UserInfo"/>
    <String value="MailCheck"/>
    <String value="Bookmark"/>
    <String value="App"/>
</Collection>
<Collection name="Sample Channels">
    <String value="SampleSimpleWebService"/>
    <String value="SampleJSP"/>
    <String value="SampleXML"/>
    <String value="SampleURLScraper"/>
</Collection>
<Collection name="News Channels">
    <String value="SampleRSS"/>
    <String value="Postit"/>
</Collection>
<Collection name="Search Channels">
    <String value="Search"/>
</Collection>
<Boolean name="defaultChannelIsMinimizable" value="true"/>
<Boolean name="defaultChannelIsMinimized" value="false"
    advanced="true"/>
<Boolean name="defaultChannelIsDetached" value="false"
    advanced="true"/>
<Boolean name="defaultChannelIsDetachable" value="true"/>
<Boolean name="defaultChannelIsRemovable" value="true"/>
<Boolean name="defaultChannelHasFrame" value="true"
    advanced="true"/>
<Boolean name="defaultChannelIsMovable" value="true"/>
<Boolean name="defaultBorderlessChannel" value="false"
    advanced="true"/>
<String name="defaultChannelColumn" value="1" advanced="true"/>
<String name="defaultChannelRow" value="1" advanced="true"/>
<Collection name="channelsIsMinimized" advanced="true"/>
<Collection name="channelsIsDetached" advanced="true"/>
```

```
        <Collection name="channelsHasFrame" advanced="true"/>
        <Collection name="channelsIsMinimizable"/>
        <Collection name="channelsRow" advanced="true"/>
        <Collection name="channelsColumn" advanced="true"/>
        <Collection name="channelsIsMovable"/>
        <Collection name="channelsIsDetachable"/>
        <Collection name="channelsIsRemovable"/>
        <Collection name="borderlessChannels"/>
    </Properties>
</Provider>
```

`CustomTCChannel.xml` File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Container name="CustomTableContainer" provider="CustomTableContainerProvider">
    <Properties>
        <String name="title" value="Front Table Container Channel"/>
        <String name="contentPage" value="toptable.jsp"/>
        <String name="description" value="This is a test for front table containers" />
        <String name="Desktop-fontFace1" value="Sans-serif"/>
        <Collection name="categories">
            <String value="Personal Channels"/>
            <String value="Sample Channels"/>
            <String value="News Channels"/>
        </Collection>
        <Collection name="Personal Channels">
            <String value="UserInfo"/>
            <String value="MailCheck"/>
            <String value="Bookmark"/>
            <String value="App"/>
        </Collection>
        <Collection name="Sample Channels">
            <String value="SampleJSP"/>
            <String value="SampleXML"/>
        </Collection>
        <Collection name="News Channels">
            <String value="SampleRSS"/>
        </Collection>
        <Collection name="channelsRow" advanced="true">
            <String name="MailCheck" value="4"/>
            <String name="App" value="5"/>
            <String name="Bookmark" value="3"/>
            <String name="SampleRSS" value="2"/>
            <String name="SampleXML" value="2"/>
        </Collection>
        <Collection name="channelsIsRemovable" >
            <Boolean name="UserInfo" value="false"/>
```

```
            </Collection>
            <Collection name="amList">
                <String value="UserInfo"/>
                <String value="MailCheck"/>
                <String value="App"/>
                <String value="Bookmark"/>
            </Collection>
            <Collection name="pmList">
                <String value="MyFrontPageTabPanelContainer/Bookmark2"/>
                <String value="MailCheck"/>
                <String value="SampleXML"/>
                <String value="SampleRSS"/>
            </Collection>
        </Properties>
        <Available>
            <Reference value="UserInfo"/>
            <Reference value="MailCheck"/>
            <Reference value="App"/>
            <Reference value="Bookmark"/>
            <Reference value="MyFrontPageTabPanelContainer/Bookmark2"/>
            <Reference value="SampleJSP"/>
            <Reference value="SampleRSS"/>
            <Reference value="SampleXML"/>
        </Available>
        <Selected>
            <Reference value="UserInfo"/>
            <Reference value="MailCheck"/>
            <Reference value="App"/>
            <Reference value="Bookmark"/>
            <Reference value="MyFrontPageTabPanelContainer/Bookmark2"/>
            <Reference value="SampleJSP"/>
            <Reference value="SampleRSS"/>
            <Reference value="SampleXML"/>
        </Selected>
        <Channels>
        </Channels>
    </Container>
```

4 Use the `psadmin` command to upload the display profile fragments for this provider.

For CustomTableContainerProvider, use the `psadmin` `add-display-profile` sub command to upload the `CustomTCProvider.xml` file and `CustomTCChannel.xml` file fragments in the display profile. See the *Sun Java System Portal Server 7.1 Command Line Reference* for more information on this sub command.

5 **Create a new directory for the provider in** *PortalServer-DataDir*`/portals/`*portal—ID*`/desktop/`*desktoptype* **directory. The directory for the provider is typically named after the channel.**

For the sample CustomTableContainerProvider, create a directory called `CustomTableContainer` in *PortalServer-DataDir*`/portals/`*portal—ID*`/desktop/`*desktoptype* directory.

6 **Develop and copy the JSP files for the provider in the newly created directory.**

For CustomTableContainerProvider, copy files from *PortalServer-DataDir*`/portals/`*portal—ID*`/desktop/default/JSPTableContainerProvider` to the *PortalServer-DataDir*`/portals/`*portal—ID*`/desktop/`*desktoptype*`/CustomTableContainer` directory.

7 **Copy the resource file for the provider into** *PortalServer-DataDir*`/portals/`*portal—ID*`/desktop/classes` **directory.**

For example, for the CustomTableContainerProvider:

a. **Change directories to** *PortalServer-DataDir*`/portals/`*portal—ID*`/desktop/classes` **directory.**

b. **Type the following command.**

```
cp JSPTableContainerProvider.properties PortalServer-DataDir
/portals/portal—ID/desktop/classes/CustomTableContainerProvider.properties
```

8 **Login to the Desktop and specify the URL in your browser to access CustomTableContainerProvider.**

`http://`*hostname*`:`*port*`/`*portal-ID*`/dt?provider=CustomTableContainer`

Example 2: Developing SpanTableContainerProvider

The sample SpanTableContainerProvider provides a layout for a `Thin_Wide_Thin_Span` column. This table Desktop layout has three columns with the `full_bottom` spanning two (left and center) columns.

▼ To Develop the SpanTableContainerProvider

1 **Extend JSPTableContainerProvider and develop the SpanTableContainerProvider class file.**

The SpanTableContainerProvider class is implemented by extending the JSPTableContainerProvider class as shown below.

`SpanTableContainerProvider.java` File

```
package custom;

import com.sun.portal.providers.containers.jsp.table.
JSPTableContainerProvider;
import com.sun.portal.providers.ProviderException;
import com.sun.portal.providers.ProviderWidths;
import com.sun.portal.providers.Provider;

import com.sun.portal.providers.context.ContainerProviderContext;

import javax.servlet.http.HttpServletRequest;

import java.util.List;

public class SpanTableContainerProvider extends
JSPTableContainerProvider {
    public int getRowSpan(HttpServletRequest req) throws
        ProviderException {
        int rowSpan = 1;
        List selected = getSelectedChannels();
        for (int i=0; i < selected.size(); i++) {
            String channel = (String)selected.get(i);
            Provider p = getContainerProviderContext().getProvider
                    (req, getName(), channel);

            if ( (p.getWidth() == ProviderWidths.WIDTH_FULL_BOTTOM) ) {
                rowSpan = rowSpan++;
            }
        }
        return rowSpan;
    }
}
```

2 Compile the class and put it in the provider class base directory.

The default directory for the class file is *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes`. That is, build the JAR and copy the JAR into the provider class base directory. Or, just copy the class as a file into the provider class base directory. To compile the `SpanTableContainerProvider.java` file, enter:

`javac -d` *PortalServer-DataDir*`/portals/`
portal-ID`/desktop/classes -classpath` *PortalServer-base*
`/sdk/desktop/desktopsdk.jar:`*AccessManager-base*
`/lib/servlet.jar SpanTableContainerProvider.java`

3 Develop the display profile XML fragments for this provider and this provider's channel.

The display profile fragment for SpanTableContainerProvider's provider is saved in `SpanTCProvider.xml` file and the display profile fragment for SpanTableContainerProvider's channel is saved in `SpanTCChannel.xml` file.

`SpanTCProvider.xml` File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Provider name="SpanTableContainerProvider" class="custom.SpanTableContainerProvider">
    <Properties>
        <String name="contentPage" value="toptable.jsp"/>
        <Integer name="timeout" value ="1800"/>
        <Integer name="layout" value="1"/>
        <Boolean name="showExceptions" value="false"/>
        <Boolean name="parallelChannelsInit" value="false"/>
        <String name="title" value="Table Container Provider"/>
        <String name="description" value="DESCRIPTION"/>
        <String name="refreshTime" value="0"/>
        <String name="width" value="thin"/>
        <String name="fontFace1" value="Sans-serif"/>
        <String name="Desktop-fontFace1" value="Sans-serif"/>
        <String name="productName" value="Sun Java System Portal Server"/>
        <String name="presetThemeChannel" value="JSPPresetThemeContainer"/>
        <String name="customThemeChannel" value="JSPCustomThemeContainer"/>
        <Boolean name="refreshParentContainerOnly" value="false" advanced="true"/>
        <Boolean name="isEditable" value="true"/>
        <String name="editType" value="edit_complete"/>
        <String name="editContainerName" value="JSPEditChannel"/>
        <Integer name="thin_popup_height" value="200"/>
        <Integer name="thin_popup_width" value="500"/>
        <Integer name="thick_popup_height" value="300"/>
        <Integer name="thick_popup_width" value="600"/>
        <Integer name="fullwidth_popup_height" value="500"/>
        <Integer name="fullwidth_popup_width" value="600"/>
        <Collection name="categories">
            <String value="Personal Channels"/>
            <String value="Sample Channels"/>
            <String value="News Channels"/>
            <String value="Search Channels"/>
        </Collection>
        <Collection name="Personal Channels">
            <String value="UserInfo"/>
            <String value="MailCheck"/>
            <String value="Bookmark"/>
            <String value="App"/>
        </Collection>
        <Collection name="Sample Channels">
```

```
            <String value="SampleSimpleWebService"/>
            <String value="SampleJSP"/>
            <String value="SampleXML"/>
            <String value="SampleURLScraper"/>
        </Collection>
        <Collection name="News Channels">
            <String value="SampleRSS"/>
            <String value="Postit"/>
        </Collection>
        <Collection name="Search Channels">
            <String value="Search"/>
        </Collection>
        <Boolean name="defaultChannelIsMinimizable" value="true"/>
        <Boolean name="defaultChannelIsMinimized" value="false" advanced="true"/>
        <Boolean name="defaultChannelIsDetached" value="false" advanced="true"/>
        <Boolean name="defaultChannelIsDetachable" value="true"/>
        <Boolean name="defaultChannelIsRemovable" value="true"/>
        <Boolean name="defaultChannelHasFrame" value="true" advanced="true"/>
        <Boolean name="defaultChannelIsMovable" value="true"/>
        <Boolean name="defaultBorderlessChannel" value="false" advanced="true"/>
        <String name="defaultChannelColumn" value="1" advanced="true"/>
        <String name="defaultChannelRow" value="1" advanced="true"/>
        <Collection name="channelsIsMinimized" advanced="true"/>
        <Collection name="channelsIsDetached" advanced="true"/>
        <Collection name="channelsHasFrame" advanced="true"/>
        <Collection name="channelsIsMinimizable"/>
        <Collection name="channelsRow" advanced="true"/>
        <Collection name="channelsColumn" advanced="true"/>
        <Collection name="channelsIsMovable"/>
        <Collection name="channelsIsDetachable"/>
        <Collection name="channelsIsRemovable"/>
        <Collection name="borderlessChannels"/>
    </Properties>
</Provider>
```

SpanTCChannel.xml File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Container name="SpanTableContainerChannel" provider="
SpanTableContainerProvider">
    <Properties>
        <String name="title" value="Front Table Container Channel"/>
        <String name="contentPage" value="toptable.jsp"/>
        <String name="description" value="This is a test for
            spanning table containers" />
        <String name="Desktop-fontFace1" value="Sans-serif"/>
        <Collection name="categories">
```

```
            <String value="Personal Channels"/>
            <String value="Sample Channels"/>
            <String value="News Channels"/>
        </Collection>
        <Collection name="Personal Channels">
            <String value="UserInfo"/>
            <String value="MailCheck"/>
            <String value="Bookmark"/>
            <String value="App"/>
        </Collection>
        <Collection name="Sample Channels">
            <String value="SampleJSP"/>
            <String value="SampleXML"/>
        </Collection>
        <Collection name="News Channels">
            <String value="SampleRSS"/>
        </Collection>
        <Collection name="channelsRow" advanced="true">
            <String name="MailCheck" value="4"/>
            <String name="App" value="5"/>
            <String name="Bookmark" value="3"/>
            <String name="SampleRSS" value="2"/>
            <String name="SampleXML" value="2"/>
        </Collection>
        <Collection name="channelsIsRemovable" >
            <Boolean name="UserInfo" value="false"/>
        </Collection>
        <Collection name="amList">
            <String value="UserInfo"/>
            <String value="MailCheck"/>
            <String value="App"/>
            <String value="Bookmark"/>
        </Collection>
        <Collection name="pmList">
            <String value="MyFrontPageTabPanelContainer
                    /Bookmark2"/>
            <String value="MailCheck"/>
            <String value="SampleXML"/>
            <String value="SampleRSS"/>
        </Collection>
    </Properties>
    <Available>
        <Reference value="UserInfo"/>
        <Reference value="MailCheck"/>
        <Reference value="App"/>
        <Reference value="Bookmark"/>
        <Reference value="MyFrontPageTabPanelContainer/
                Bookmark2"/>
```

```
            <Reference value="SampleJSP"/>
            <Reference value="SampleRSS"/>
            <Reference value="SampleXML"/>
        </Available>
        <Selected>
            <Reference value="UserInfo"/>
            <Reference value="MailCheck"/>
            <Reference value="App"/>
            <Reference value="Bookmark"/>
            <Reference value="MyFrontPageTabPanelContainer/
                    Bookmark2"/>
            <Reference value="SampleJSP"/>
            <Reference value="SampleRSS"/>
            <Reference value="SampleXML"/>
        </Selected>
        <Channels>
        </Channels>
    </Container>
```

4 Use the `psadmin` command to upload the display profile fragments for this provider.

For SpanTableContainerProvider, use the `psadmin add-display-profile` sub command to upload the `CustomTCProvider.xml` file and `CustomTCChannel.xml` file fragments in the display profile. See the *Sun Java System Portal Server 7.1 Command Line Reference* for more information on this sub command.

5 Create a new directory for the provider in *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype* directory. The directory for the provider is typically named after the channel.

For the sample SpanTableContainerProvider, create a directory called `SpanTableContainerChannel` in *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype* directory.

6 Copy files from *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/default/JSPTableContainerProvider` to the newly created directory.

7 Modify the JSP files (in *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype*`/SpanTableContainerChannel`) as shown in Step a, Step b, and Step c (modifications to the file are shown in bold):

a. Add the code to the `toptable.jsp` file:

```
<!-- BEGIN FULL TOP CHANNELS -->
<dttable:getColumns id="channels" column="top" scope="request"/>
<jsp:include page="tabletopbottom.jsp" flush="true">
<jsp:param name="columnName" value="top"/>
```

```
</jsp:include>
<!-- END TOP CHANNELS -->
<!-- BEGIN FULL BOTTOM CHANNELS -->
<dttable:getColumns id="channels" column="bottom" scope="request"/>
<jsp:include page="tabletopbottom.jsp" flush="true">
<jsp:param name="columnName" value="bottom"/>
</jsp:include>
<!-- END BOTTOM CHANNELS -->
```

Here, `tcp` is an instance of SpanTableContainerProvider.

b. Add the code to the `tablecolumn.jsp` file:

```
<%@ page import="custom.SpanTableContainerProvider"%>
<%@ page session="false" %>
<% SpanTableContainerProvider tcp =
(SpanTableContainerProvider)pageContext.getAttribute("JSPProvider"); %>
<dt:obtainContainer container="$JSPProvider">
<% String columnName = request.getParameter( "columnName" ); %>
<% pageContext.setAttribute( "columnName", columnName ); %>
<% if (columnName.equalsIgnoreCase("right")) { %>
<TD WIDTH="<dttable:getColumnWidth column="$columnName"/>%"
VALIGN=TOP ROWSPAN="<%=tcp.getRowSpan(request)%>"> <% } else { %>
<TD WIDTH="<dttable:getColumnWidth column="$columnName"/>%" VALIGN=TOP>
<% } %>
```

c. Add the code to the `tabletopbottom.jsp` file:

```
<jx:if test="$hasChannel">
<TABLE WIDTH=100% BORDER=0 CELLPADDING=2 CELLSPACING=0>
<% if (request.getParameter("columnName").equals("bottom")) { %>
<TR COLSPAN=2>
<% } else { %>
<TR>
<% } %>
<TD WIDTH="100%" VALIGN=TOP>
<jx:forEach var="channel" items="$channels">
```

8 Copy the *PortalServer-DataDir*/portals/*portal-ID*/desktop/classes/JSPTableContainerProvider.properties to *PortalServer-DataDir*/portals/*portal-ID*/desktop/classes/SpanTableContainerProvider.properties file.

9 At least one of the selected channels should be of width `full_bottom` and the selected layout should be `thin_wide_thin` for the sample layout to be displayed.

10 Specify the URL in your browser to access SpanTableContainerProvider.

`http://`*hostname*`:`*port*`/`*portal-ID*`/dt?provider=SpanTableContainerChannel`

Customizing the Row Layout

This section provides some sample extensions to the JSPTableContainerProvider class file and its associated JSP files for enabling a customized Desktop layout.

▼ To Develop the CustomTCProvider

1 Create a new custom table container provider by extending JSPTableContainerProvider.

Override the `getLayout()`, `setUpColumns()`, and `getWidths()` methods as shown in the `CustomTCProvider.java` file. You can also add any new logic to arrange the channels into rows and columns according to your new layout.

`CustomTCProvider.java` File

```
package custom;

import com.sun.portal.providers.containers.jsp.table.
JSPTableContainerProvider;
import com.sun.portal.providers.ProviderException;
import com.sun.portal.providers.ProviderWidths;
import com.sun.portal.providers.Provider;
import com.sun.portal.providers.util.Layout;

import com.sun.portal.providers.context.
ContainerProviderContext;

import javax.servlet.http.HttpServletRequest;

import java.util.List;
import java.util.ArrayList;
import java.util.Map;
import java.util.HashMap;
import java.util.Iterator;

public class CustomTCProvider extends JSPTableContainerProvider {
    public int getLayout() throws ProviderException {
        int layout = getIntegerProperty("layout");
        return layout;
    }

    public Map setupColumns(HttpServletRequest req)
        throws ProviderException {
        int layout = getLayout();
        List leftList = new ArrayList();
        List centerList = new ArrayList();
        List rightList = new ArrayList();
        List fullTopList = new ArrayList();
```

```
        List fullBottomList = new ArrayList();
        if ((layout >= Layout.LAYOUT_THIN_THICK) && (layout <=
            Layout.LAYOUT_THIN_THIN_THIN)) {
            return super.setupColumns(req);
        } else {
            List selectedChannels = getSelectedChannels();
            for (Iterator i = selectedChannels.iterator(); i.hasNext(); ) {
                String providerName = (String)i.next();
                Provider p = null;
                p = getContainerProviderContext().getProvider(req,
getName(), providerName);
                int width = -1;
                try {
                    width = p.getWidth();
                }
                  catch (NumberFormatException e) {
                }
                  catch (Throwable e) {
                }
            }
            // add the channel to leftList/centerList/rightList/
                fullTopList/fullBottomList based on width and as required by
                your new layout.
        }
        Map<Integer,List> columnMap = new HashMap<Integer,List>();

        columnMap.put( new Integer(LEFT), leftList );
        columnMap.put( new Integer(RIGHT), rightList );
        columnMap.put( new Integer(CENTER), centerList );
        columnMap.put( new Integer(TOP), fullTopList );
        columnMap.put( new Integer(BOTTOM), fullBottomList );
        return columnMap;
    }
    public int getWidths(int column) throws ProviderException {
        int centerWidth = -1;
        int leftWidth = -1;
        int rightWidth = -1;
        /* modify this method as shown in the customizing column widths
section according to your layout specifications or keep this method as is.*/
        //return the corresponding width based on the column.
        switch( column ) {
            case LEFT:
            return leftWidth;
            case RIGHT:
            return rightWidth;
            case CENTER:
            return centerWidth;
            default:
```

```
            return -1;
        }
    }
}
```

2 **Compile the class and put it in the provider class base directory.**

The default directory for the class file is *PortalServer-DataDir*/portals/*portal-ID*/desktop/classes/. That is, build the JAR and copy the JAR into the provider class base directory. Or, just copy the class as a file into the provider class base directory. To compile the CustomTCProvider.java file, enter:

```
javac -d PortalServer-DataDir/portals/portal-ID
/desktop/classes/ -classpath PortalServer-base/sdk/desktop/desktopsdk.jar:
AccessManager-base/lib/servlet.jar CustomTCProvider.java
```

3 **Develop the display profile XML fragment for this provider and this provider's channel and use the** psadmin **command to upload the display profile fragments for this provider.**

a. **The display profile fragment for the CustomTCProvider's provider and channel must be saved in separate XML files.**

b. **Add** <Integer name="layout" value="5"/> **in the display profile fragment for the CustomTCProvider's provider.**

c. **Use the** psadmin add-display-profile **sub command to upload the** CustomTCProvider.xml **file and** CustomTCChannel.xml **file fragments in the display profile.**

See the *Sun Java System Portal Server 7.1 Command Line Reference* for more information on this sub command.

4 **Create a new directory for the provider in** *PortalServer-DataDir*/portals/*portal-ID*/desktop/*desktoptype* **directory and copy files from** *PortalServer-DataDir*/portals/*portal-ID*/desktop/default/JSPTableContainerProvider **to the newly created directory.**

The directory for the provider is typically named after the provider. For the sample CustomTCProvider, create a directory called CustomTCProvider in *PortalServer-DataDir*/portals/*portal-ID*/desktop/*desktoptype* directory.

5 **Modify the HTML in** toptable.jsp **file in** *PortalServer-DataDir*/portals/*portal-ID*/desktop/*desktoptype*/CustomTCProvider**.**

Rearrange the channels into rows and columns as per your specifications. Replace JSPLayoutContainer with CustomLayoutContainer.

6 **Create a new directory for CustomLayoutContainer under** *PortalServer-DataDir*/portals/*portal-ID*/desktop/*desktoptype* **directory and copy files from**

PortalServer-DataDir/portals/*portal-ID*/desktop/default/JSPLayoutContainer **to** *PortalServer-DataDir*/portals/*portal-ID*/desktop/*desktoptype*/CustomLayoutContainer **directory.**

For example, type:

```
mkdir PortalServer-DataDir/portals/portal-ID/desktop/
desktoptype/CustomLayoutContainer
cp PortalServer-DataDir/portals/
portal-ID/desktop/default/JSPLayoutContainer/*
PortalServer-DataDir/portals/portal-ID
/desktop/desktoptype/CustomLayoutContainer
```

7 Modify `layoutedit.jsp` **to display your new layout in the layout page.**

Modify the `layoutedit.jsp` file at *PortalServer-DataDir*/portals/*portal-ID*/desktop/*desktoptype*/CustomLayoutContainer.

`layoutedit.jsp` File

```
<jx:choose>
    <jx:when test="$layout == 1">
        <%@ include file="layout1.jsp" %>
    </jx:when>
    <jx:when test="$layout == 2">
        <%@ include file="layout2.jsp" %>
    </jx:when>
    <jx:when test="$layout == 3">
        <%@ include file="layout3.jsp" %>
    </jx:when>
    <jx:when test="$layout == 5">
        <%@ include file="layout5.jsp" %>
    </jx:when>
</jx:choose>
```

8 Create a new `layout5.jsp` **which gets displayed on the layout page when the user selects layout 5 and modify** `layoutdoedit.jsp` **to handle the processing of layout 5 that you have created.**

9 Add the display profile for CustomLayoutContainer as shown in `CustomLayoutContainer.xml` **file and upload the display profile XML fragment for CustomLayoutContainer using the** `psadmin` **command.**

`CustomLayoutContainer.xml` File

```
<Container name="CustomLayoutContainer" provider=
"JSPSingleContainerProvider" advanced="true">
    <Properties>
        <String name="title" value="JSP Custom Layout Channel"/>
        <String name="description" value="This is the JSP Layout Channel"/>
        <String name="contentPage" value="" merge="replace"
            lock="false" propagate="false"/>
```

```
        <String name="editPage" value="layoutedit.jsp"/>
        <String name="processPage" value="layoutdoedit.jsp"/>
        <Boolean name="isEditable" value="true" advanced="true"/>
        <String name="editType" value="edit_complete" advanced="true"/>
    </Properties>

    <Available>
    </Available>
    <Selected>
    </Selected>
    <Channels>
    </Channels>
</Container>
```

To upload `CustomLayoutContainer.xml` file fragments, use the `psadmin add-display-profile` sub command. See the *Sun Java System Portal Server 7.1 Command Line Reference* for more information on this sub command.

10 Specify the URL in your browser to access CustomTCProvider.

`http://`*hostname*`:`*port*`/`*portal-ID*`/dt?provider=CustomTCProvider`

Customizing the Column Widths

The `getWidths(int column)` method throws ProviderException; it returns the column width for the column the width is requested for. By default, the column widths for left, right columns in layout 1 and layout 2 are defined as 30, 70 and for layout 3 they are defined as 25, 50, 25.

To customize the column widths, extend the JSPTableContainerProvider and override the `getWidths(int column)` method to return different widths.

▼ To Develop the CustomJSPTableContainerProvider

1 Extend the JSPTableContainerProvider and develop the CustomJSPTableContainerProvider class file.

The CustomJSPTableContainerProvider class file in `CustomJSPTableContainerProvider.java` file overrides the `getWidths()` method in the JSPTableContainerProvider and returns 40 for left, and 60 for right columns, and 30, 40, 30 for layout 3 (changes are shown in bold).

`CustomJSPTableContainerProvider.java` File

```
package custom;

import com.sun.portal.providers.containers.jsp.table.
JSPTableContainerProvider;
import com.sun.portal.providers.ProviderException;
```

```
import com.sun.portal.providers.ProviderWidths;
import com.sun.portal.providers.Provider;
import com.sun.portal.providers.util.Layout;

public class CustomJSPTableContainerProvider extends
JSPTableContainerProvider {
    public int getWidths(int column) throws ProviderException {
        int centerWidth = -1;
        int rightWidth = -1;
        int leftWidth = -1;
        int layout = getLayout();
        switch (layout) {
            case Layout.LAYOUT_THIN_THICK:
                leftWidth = 40;
                rightWidth = 60;
                break;
            case Layout.LAYOUT_THICK_THIN:
                rightWidth = 40;
                leftWidth = 60;
                break;
            case Layout.LAYOUT_THIN_THICK_THIN:
                rightWidth = 30;
                centerWidth= 40;
                leftWidth = 30;
                break;
            default:
                rightWidth = 40;
                leftWidth = 60;
                break;
        }
        switch( column ) {
            case LEFT:
                return leftWidth;
            case RIGHT:
                return rightWidth;
            case CENTER:
                return centerWidth;
            default:
                return -1;
        }
    }
}
```

2 Compile the class and put it in the provider class base directory.

The default directory for the class file is *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes/`. That is, build the JAR and copy

the JAR into the provider class base directory. Or, just copy the class as a file into the provider class base directory. To compile the `CustomJSPTableContainerProvider.java` file, enter:

```
javac -d PortalServer-DataDir/portals/
portal-ID/desktop/classes/ -classpath BaseDir/SUNWps/sdk/desktop
/desktopsdk.jar:BaseDir/SUNWam/lib/servlet.jar
CustomJSPTableContainerProvider.java
```

3 Develop the display profile XML fragments for this provider and this provider's channel.

The display profile fragment for the CustomJSPTableContainerProvider's provider is saved in `CustomJSPTCProvider.xml` file and the display profile fragment for the CustomTableContainerProvider's channel is saved in `CustomJSPTCChannel.xml` file.

`CustomJSPTCProvider.xml` File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Provider name="CustomJSPTableContainerProvider" class="
custom.CustomJSPTableContainerProvider">
    <Properties>
        <String name="contentPage" value="toptable.jsp"/>
        <Integer name="timeout" value ="1800"/>
        <Integer name="layout" value="1"/>
        <Boolean name="showExceptions" value="false"/>
        <Boolean name="parallelChannelsInit" value="false"/>
        <String name="title" value="Table Container Provider"/>
        <String name="description" value="DESCRIPTION"/>
        <String name="refreshTime" value="0"/>
        <String name="width" value="thin"/>
        <String name="fontFace1" value="Sans-serif"/>
        <String name="Desktop-fontFace1" value="Sans-serif"/>
        <String name="productName" value="Sun Java System Portal Server"/>
        <String name="presetThemeChannel" value="JSPPresetThemeContainer"/>
        <String name="customThemeChannel" value="JSPCustomThemeContainer"/>
        <Boolean name="refreshParentContainerOnly" value="false" advanced="true"/>
        <Boolean name="isEditable" value="true"/>
        <String name="editType" value="edit_complete"/>
        <String name="editContainerName" value="JSPEditChannel"/>
        <Integer name="thin_popup_height" value="200"/>
        <Integer name="thin_popup_width" value="500"/>
        <Integer name="thick_popup_height" value="300"/>
        <Integer name="thick_popup_width" value="600"/>
        <Integer name="fullwidth_popup_height" value="500"/>
        <Integer name="fullwidth_popup_width" value="600"/>
        <Collection name="categories">
            <String value="Personal Channels"/>
            <String value="Sample Channels"/>
            <String value="News Channels"/>
```

```
                <String value="Search Channels"/>
            </Collection>
            <Collection name="Personal Channels">
                <String value="UserInfo"/>
                <String value="MailCheck"/>
                <String value="Bookmark"/>
                <String value="App"/>
            </Collection>
            <Collection name="Sample Channels">
                <String value="SampleSimpleWebService"/>
                <String value="SampleJSP"/>
                <String value="SampleXML"/>
                <String value="SampleURLScraper"/>
            </Collection>
            <Collection name="News Channels">
                <String value="SampleRSS"/>
                <String value="Postit"/>
            </Collection>
            <Collection name="Search Channels">
                <String value="Search"/>
            </Collection>
            <Boolean name="defaultChannelIsMinimizable" value="true"/>
            <Boolean name="defaultChannelIsMinimized" value="false"
                advanced="true"/>
            <Boolean name="defaultChannelIsDetached" value="false"
                advanced="true"/>
            <Boolean name="defaultChannelIsDetachable" value="true"/>
            <Boolean name="defaultChannelIsRemovable" value="true"/>
            <Boolean name="defaultChannelHasFrame" value="true"
                advanced="true"/>
            <Boolean name="defaultChannelIsMovable" value="true"/>
            <Boolean name="defaultBorderlessChannel" value="false"
                advanced="true"/>
            <String name="defaultChannelColumn" value="1" advanced="true"/>
            <String name="defaultChannelRow" value="1" advanced="true"/>
            <Collection name="channelsIsMinimized" advanced="true"/>
            <Collection name="channelsIsDetached" advanced="true"/>
            <Collection name="channelsHasFrame" advanced="true"/>
            <Collection name="channelsIsMinimizable"/>
            <Collection name="channelsRow" advanced="true"/>
            <Collection name="channelsColumn" advanced="true"/>
            <Collection name="channelsIsMovable"/>
            <Collection name="channelsIsDetachable"/>
            <Collection name="channelsIsRemovable"/>
            <Collection name="borderlessChannels"/>
        </Properties>
    </Provider>
```

CustomJSPTCChannel.xml File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Container name="CustomJSPTableContainerChannel"
provider="CustomJSPTableContainerProvider">
    <Properties>
        <String name="title" value="Front Table Container Channel"/>
        <String name="contentPage" value="toptable.jsp"/>
        <String name="description" value="This is a test for
            front table containers" />
        <String name="Desktop-fontFace1" value="Sans-serif"/>
        <Collection name="categories">
            <String value="Personal Channels"/>
            <String value="Sample Channels"/>
            <String value="News Channels"/>
        </Collection>
        <Collection name="Personal Channels">
            <String value="UserInfo"/>
            <String value="MailCheck"/>
            <String value="Bookmark"/>
            <String value="App"/>
        </Collection>
        <Collection name="Sample Channels">
            <String value="SampleJSP"/>
            <String value="SampleXML"/>
        </Collection>
        <Collection name="News Channels">
            <String value="SampleRSS"/>
        </Collection>
        <Collection name="channelsRow" advanced="true">
            <String name="MailCheck" value="4"/>
            <String name="App" value="5"/>
            <String name="Bookmark" value="3"/>
            <String name="SampleRSS" value="2"/>
            <String name="SampleXML" value="2"/>
        </Collection>
        <Collection name="channelsIsRemovable" >
            <Boolean name="UserInfo" value="false"/>
        </Collection>
        <Collection name="amList">
            <String value="UserInfo"/>
            <String value="MailCheck"/>
            <String value="App"/>
            <String value="Bookmark"/>
        </Collection>
        <Collection name="pmList">
```

```
                <String value="MyFrontPageTabPanelContainer/
                    Bookmark2"/>
                <String value="MailCheck"/>
                <String value="SampleXML"/>
                <String value="SampleRSS"/>
            </Collection>
        </Properties>
        <Available>
            <Reference value="UserInfo"/>
            <Reference value="MailCheck"/>
            <Reference value="App"/>
            <Reference value="Bookmark"/>
            <Reference value="MyFrontPageTabPanelContainer/
                Bookmark2"/>
            <Reference value="SampleJSP"/>
            <Reference value="SampleRSS"/>
            <Reference value="SampleXML"/>
        </Available>
        <Selected>
            <Reference value="UserInfo"/>
            <Reference value="MailCheck"/>
            <Reference value="App"/>
            <Reference value="Bookmark"/>
            <Reference value="MyFrontPageTabPanelContainer/
                Bookmark2"/>
            <Reference value="SampleJSP"/>
            <Reference value="SampleRSS"/>
            <Reference value="SampleXML"/>
        </Selected>
        <Channels>
        </Channels>
    </Container>
```

4 Use the `psadmin` command to upload the display profile fragments for this provider.

For CustomJSPTableContainerProvider, use the `psadmin add-display-profile` sub command to upload the `CustomTCProvider.xml` file and `CustomTCChannel.xml` file fragments in the display profile. See the *Sun Java System Portal Server 7.1 Command Line Reference* for more information on this sub command.

5 Create a new directory for the provider in *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype* directory. The directory for the provider is typically named after the provider.

For the sample CustomJSPTableContainerProvider, create a directory called `CustomJSPTableContainerChannel` in *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype* directory.

6. **Develop and copy the JSP files for the provider in the newly created directory.**

   For CustomJSPTableContainerProvider, copy files from *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/default/JSPTableContainerProvider` to the *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype*`/CustomJSPTableContainerChannel` directory.

7. **Copy the resource file for the provider into** *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes` **directory.**

   For example, for the CustomJSPTableContainerProvider:

   a. **Change directories to** *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes` **directory.**

   b. **Type** `cp JSPTableContainerProvider.properties` *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes/CustomJSPTableContainerProvider`

8. **Login to the Desktop and specify the URL in your browser to access CustomJSPTableContainerProvider.**

   `http://`*hostname*`:`*port*`/`*portal-ID*`/dt?provider=CustomJSPTableContainerChannel`

Extending the JSPTabContainerProvider

This section includes a sample extension to the JSPTabContainerProvider class. In this example, a new method called `getTabTopics()` is added to the JSPTabContainerProvider. The `getTabTopics()` method returns only the predefined tabs from the list of available tabs. This method is then called from the `makeNewTab.jsp` so that the Make New Tab page of the customTabContainer will only display the Predefined tab topics instead of displaying all the available tabs (predefined and user created).

▼ To Develop the CustomTabContainerProvider

1. **Extend the JSPTabContainerProvider and develop the CustomTabContainerProvider class file.**

   The CustomTabContainerProvider class file includes a `getTabTopics()` method that returns only the predefined tabs from the list of available tabs.

   `CustomTabContainerProvider.java` File

```
package custom;

import com.sun.portal.providers.containers.jsp.tab.JSPTabContainerProvider;
import com.sun.portal.providers.containers.jsp.tab.UnmodifiableTab;
```

```
import com.sun.portal.providers.ProviderException;

import java.util.List;
import java.util.ArrayList;

public class CustomTabContainerProvider extends JSPTabContainerProvider {
    public List getTabTopics() throws ProviderException {
        List<UnmodifiableTab> availtabs = new ArrayList<UnmodifiableTab>();
        List tabs = super.getAvailableTabs();
        for (int i=0; i < tabs.size(); i++) {
            UnmodifiableTab tab = (UnmodifiableTab)tabs.get(i);
            if (tab.isPredefined()) {
                availtabs.add(tab);
            }
        }
        return availtabs;
    }
}
```

2 Compile the class and put it in the provider class base directory.

The default directory for the class file is *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/classes`. That is, build the JAR and copy the JAR into the provider class base directory. Or, just copy the class as a file into the provider class base directory. To compile the `CustomTabContainerProvider.java` file, enter:

```
javac -d PortalServer-DataDir
/portals/portal-ID/desktop/classes -classpath
PortalServer-base/sdk/desktop
/desktopsdk.jar:AccessManager-base/lib/
servlet.jar CustomTabContainerProvider.java
```

3 Develop the display profile XML fragments for this provider and this provider's channel.

The display profile fragments for the CustomTabContainerProvider's provider and channel are saved in `CustomTabCProvider.xml` and `CustomTabCChannel.xml` files respectively.

`CustomTabCProvider.xml` File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Provider name="CustomTabContainerProvider" class="
custom.CustomTabContainerProvider">
    <Properties>
        <String name="contentPage" value="tab.jsp"/>
        <Boolean name="showExceptions" value="false"/>
        <String name="title" value="*** Tab Container Provider ***"/>
        <String name="description" value="*** DESCRIPTION ***"/>
        <String name="refreshTime" value="" advanced="true"/>
        <String name="width" value="thin" advanced="true"/>
```

```
        <String name="fontFace1" value="Sans-serif"/>
        <String name="productName" value="Sun Java System Portal Server"/>
        <String name="presetThemeChannel" value="
            JSPPresetThemeContainer" advanced="true"/>
        <String name="customThemeChannel" value="JSPCustomThemeContainer"
            advanced="true"/>
        <Boolean name="isEditable" value="true" advanced="true"/>
        <String name="editType" value="edit_complete" advanced="true"/>
        <String name="editContainerName" value="JSPEditContainer"
            advanced="true"/>
            <Integer name="thin_popup_height" value="200"/>
        <Integer name="thin_popup_width" value="500"/>
        <Integer name="thick_popup_height" value="300"/>
        <Integer name="thick_popup_width" value="600"/>
        <Integer name="fullwidth_popup_height" value="500"/>
        <Integer name="fullwidth_popup_width" value="600"/>
        <Boolean name="defaultChannelIsMinimizable" value="true"/>
        <Boolean name="defaultChannelIsMinimized" value="false"
            advanced="true"/>
        <Boolean name="defaultChannelIsDetached" value="false"
            advanced="true"/>
        <Boolean name="defaultChannelIsDetachable" value="true"/>
        <Boolean name="defaultChannelIsRemovable" value="true"/>
        <Boolean name="defaultChannelHasFrame" value="true"
            advanced="true"/>
        <Boolean name="defaultChannelIsMovable" value="true"/>
        <Boolean name="defaultBorderlessChannel" value="false"
            advanced="true"/>
        <String name="defaultChannelColumn" value="1" advanced="true"/>
        <String name="defaultChannelRow" value="1" advanced="true"/>
        <Collection name="channelsIsMinimized" advanced="true"/>
        <Collection name="channelsIsDetached" advanced="true"/>
        <Collection name="channelsHasFrame" advanced="true"/>
        <Collection name="channelsIsMinimizable"/>
        <Collection name="channelsRow" advanced="true"/>
        <Collection name="channelsColumn" advanced="true"/>
        <Collection name="channelsIsMovable"/>
        <Collection name="channelsIsDetachable"/>
        <Collection name="channelsIsRemovable"/>
        <Collection name="borderlessChannels"/>
    </Properties>
</Provider>
```

`CustomTabCChannel.xml` File

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Container name="CustomTabContainer" provider="CustomTabContainerProvider"
```

```
merge="replace" lock="false" advanced="false">
    <Properties advanced="false" merge="fuse" lock="false"
        name="_properties" propagate="true">
        <String name="title" value="JSP Tab Container Channel"
            advanced="false"
            merge="replace" lock="false" propagate="true"/>
        <String name="description" value="This is a test for tab
            containers" advanced="false" merge="replace" lock="false"
            propagate="true"/>
        <String name="contentPage" value="tab.jsp" advanced="false"
            merge="replace" lock="false" propagate="true"/>
        <String name="editPage" value="tabedit.jsp" advanced="false"
             merge="replace" lock="false" propagate="true"/>
        <String name="startTab" value="MyFrontPageTabPanelContainer"
            advanced="false" merge="replace" lock="false" propagate="true"/>
        <Integer name="maxTabs" value="4" advanced="false" merge="replace"
            lock="false" propagate="true"/>
        <String name="makeTabProvider" value="JSPTabCustomTableContainerProvider"
            advanced="true" merge="replace" lock="false" propagate="true"/>
        <String name="makeTabChannel" value="JSPTabCustomTableContainer"
            advanced="true" merge="replace" lock="false" propagate="true"/>
        <Integer name="channelNumber" value="0" advanced="false"
            merge="replace" lock="false" propagate="true"/>
        <String name="contentChannel" value="JSPContentContainer"
            advanced="false" merge="replace" lock="false" propagate="true"/>

        <Collection name="TabProperties" advanced="false" merge="fuse"
            lock="false" propagate="true">
            <Collection name="MyFrontPageTabPanelContainer" advanced="false"
                 merge="fuse" lock="false" propagate="true">
                <String name="title" value="My Front Page" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
                <String name="desc" value="Your front page" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
                <Boolean name="removable" value="false" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
                <Boolean name="renamable" value="true" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
                <Boolean name="predefined" value="true" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
            </Collection>

            <Collection name="SamplesTabPanelContainer" advanced="false"
                    merge="fuse" lock="false" propagate="true">
                <String name="title" value="Samples" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
                <String name="desc" value="Sampless Tab" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
```

```
                <Boolean name="removable" value="true" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
                <Boolean name="renamable" value="true" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
                <Boolean name="predefined" value="true" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
            </Collection>

            <Collection name="JSPTabCustomTableContainer" advanced="false"
                merge="fuse" lock="false" propagate="true">
                <String name="title" value="Make My Own Tab" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
                <String name="desc" value="Make from Scratch" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
                <Boolean name="removable" value="true" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
                <Boolean name="renamable" value="true" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
                <Boolean name="predefined" value="false" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
            </Collection>

            <Collection name="SearchTabPanelContainer" advanced="false"
                    merge="fuse" lock="false" propagate="true">
                <String name="title" value="Search" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
                <String name="desc" value="Search Tab" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
                <Boolean name="removable" value="true" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
                <Boolean name="renamable" value="true" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
                <Boolean name="predefined" value="true" advanced="false"
                      merge="replace" lock="false" propagate="true"/>
            </Collection>
        </Collection>
    </Properties>

    <Available advanced="false" merge="fuse" lock="false">
        <Reference value="MyFrontPageTabPanelContainer" advanced="false"
            merge="replace" lock="false" propagate="true"/>
        <Reference value="SamplesTabPanelContainer" advanced="false"
            merge="replace" lock="false" propagate="true"/>
        <Reference value="SearchTabPanelContainer" advanced="false"
            merge="replace" lock="false" propagate="true"/>
    </Available>

    <Selected advanced="false" merge="fuse" lock="false">
```

```
        <Reference value="MyFrontPageTabPanelContainer" advanced="false"
            merge="replace" lock="false" propagate="true"/>
        <Reference value="SamplesTabPanelContainer" advanced="false"
            merge="replace" lock="false" propagate="true"/>
        <Reference value="SearchTabPanelContainer" advanced="false"
            merge="replace" lock="false" propagate="true"/>
    </Selected>

    <Channels>
    </Channels>

</Container>
```

4 Use the `psadmin` command to upload the display profile fragments for this provider.

For CustomTabContainerProvider, use the `psadmin` `add-display-profile` sub command to upload the `CustomTabCProvider.xml` and `CustomTabCChannel.xml` file fragments in the display profile. See the *Sun Java System Portal Server 7.1 Command Line Reference* for more information on this sub command.

5 Create a new directory called `CustomTabContainerProvider` under *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype* directory.

6 Copy all the required JSP files from *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/default/JSPTabContainerProvider` to *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype*`/CustomTabContainerProvider` directory.

7 Modify the `makeNewTab.jsp` file in *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/`*desktoptype*`/CustomTabContainerProvider` to call `getTabTopics()`.

That is, replace the `makeNewTab.jsp` file with the `makeNewTab.jsp` file shown below.

`makeNewTab.jsp` File

```
<%--
Copyright 2001 Sun Microsystems, Inc.  All rights reserved.
PROPRIETARY/CONFIDENTIAL.  Use of this product is subject to
license terms.
--%>
<%-- makeNewTab.jsp --%><%@ page import="custom.
CustomTabContainerProvider" %>
<table border=0 cellspacing=2 cellpadding=2 width="50%">
    <tr>
        <td align="right"><font face="<%=fontFace1%>" size="+0">
        <b>Tab Name:</b></font></td>
        <td><font face="<%=fontFace1%>" size="+0"><input type="text"
        name="JSPTabContainer.tabName" size="30"></font></td>
```

```
    </tr>
    <tr>
        <td align="right"><font face="<%=fontFace1%>" size="+0">
        <b>Tab Description:</b></font></td>
        <td><font face="<%=fontFace1%>" size="+0"><input type="text"
        name="JSPTabContainer.tabDesc" size="30"></font></td>
    </tr>
    <tr>
        <td align="right"><font face="<%=fontFace1%>" size="+0">
        <b>Tab Topics:</b></font></td>
        <td>
            <!-- start tab topic radios -->
            <table border="0" cellpadding="2" cellspacing="2">
                <%
                    String checked = "";
                %>
                <tab:tabContainerProvider>
                <%
                    CustomTabContainerProvider ctcp =
                    (CustomTabContainerProvider)pageContext.getAttribute
                    ("JSPProvider");
                    List tabTopics = ctcp.getTabTopics();
                    for (int i=0; i < tabTopics.size(); i++) {
                    UnmodifiableTab tab = (UnmodifiableTab)tab.get(i);
                %>
                <TR>
                    <TD><INPUT TYPE="radio" NAME="JSPTabContainer.tabTopic"
                    VALUE="<%=tab.getEncodedName%>" <%=checked%> ></TD>
                    <TD><FONT SIZE="-1" FACE="sans-serif"> <B>
                    <%=tab.getDisplayname%> </B> <BR> <FONT SIZE="-1"
                    FACE="sans-serif"><%=tab.getDesc%></FONT></FONT></TD>
                </TR>
                <%
                }
                %>
                <%
                checked = "CHECKED";
                %>
                <tab:getMakeTab id="makeTab"/>
                <tab:obtainTab tab="$makeTab">
                <%@ include file="makeTopic.jsp" %>
                </tab:obtainTab>
                </tab:tabContainerProvider>
            </table>
            <!-- end tab topic radios -->
            <input type="hidden" name="JSPTabContainer.make" value="make">
        </td>
    </tr>
</table>
```

8 Copy the resource file for the provider into *PortalServer-DataDir*/portals/*portal-ID*/desktop/classes directory.

For example, for the CustomJSPTableContainerProvider:

a. Change directories to *PortalServer-DataDir*/portals/*portal-ID*/desktop/classes directory.

b. Type the following command.

```
cp JSPTabContainerProvider.properties
PortalServer-DataDir/portals/portal-ID/desktop/
classes/CustomTabContainerProvider.properties
```

9 Login to the Desktop and specify the URL in your browser to access CustomTabContainerProvider.

`http://`*hostname*`:`*port*`/`*portal-ID*`/dt?provider=CustomTabContainer`

◆ ◆ ◆ CHAPTER 10

10 Creating a Custom ContainerProvider

This chapter includes detailed instructions for creating a sample RouterContainerProvider. A RouterContainerProvider provides the interface for the user to select between a tab and table Desktop layout. Use the instructions (provided with the sample RouterContainerProvider) for developing your custom ContainerProvider by extending any one of the container providers.

Creating a Custom ContainerProvider

▼ To Develop the RouterContainerProvider

1 **Based on whether the container provider is template based or JSP based container provider, create a new class which extends ContainerProviderAdapter or JSPContainerProviderAdapter.**

The new custom container provider can directly extend the container classes. For example:

```
public class CustomContainerProvider extends *** {
    // Implement the methods
}
```

See below for the sample `RouterContainerProvider.java` file:

`RouterContainerProvider.java` File

```
package com.sample.providers.containers.router;

import java.util.List;
import java.util.ArrayList;

import java.net.URL;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import com.sun.portal.providers.containers.jsp.single.
JSPSingleContainerProvider;
import com.sun.portal.providers.ProviderException;

public class RouterContainerProvider
extends JSPSingleContainerProvider {

    public URL processEdit(HttpServletRequest req,
        HttpServletResponse res) throws ProviderException {
        String selectedDesktop = req.getParameter("desktop");
        setSelectedChannel(selectedDesktop);
        return null;
    }

    /**
    * Sets the selected channel name.
    *
    * @param the selected channel name <code>String</code>.
    *
    * @exception ProviderException if the selected channel
        cannot be set
    **/
    public void setSelectedChannel(String channel) throws
        ProviderException {
        List<String> selectedChannels = new ArrayList<String>();
        selectedChannels.add(channel);
        setSelectedChannels(selectedChannels);
    }
}
```

2 Compile the class file and put it in the provider class base directory.

To compile the sample RouterContainerProvider, type:

```
javac -d PortalServer-DataDir/portals/
portal-ID/desktop/classes -classpath
PortalServer-base/sdk/desktop/desktopsdk.jar:
AccessManager-base/lib/servlet.jar RouterContainerProvider.java
```

3 Define the ContainerProvider provider and container channel XML fragments in the display profile and use the `psadmin` command to upload the display profile fragments for this provider.

The sample RouterContainerProvider provider XML fragment is in the `RCProvider.xml` file.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

<Provider name="RouterContainerProvider" class="com.sample.
providers.containers.router.RouterContainerProvider">
    <Properties>
```

```
        <String name="contentPage" value="router.jsp"/>
        <String name="editPage" value="routeredit.jsp"/>
        <String name="processPage" value=""/>
        <Boolean name="showExceptions" value="false"/>

        <String name="title" value="*** Router Container
            Provider ***"/>
        <String name="description" value="*** DESCRIPTION ***"/>
        <String name="refreshTime" value="" advanced="true"/>
        <String name="width" value="thin" advanced="true"/>
        <String name="fontFace1" value="Sans-serif"/>
        <String name="productName" value="Sun Java System Portal Server"/>

        <Boolean name="isEditable" value="true" advanced="true"/>
        <String name="editType" value="edit_complete" advanced="true"/>
        <String name="editContainerName" value="JSPEditContainer"
            advanced="true"/>

        <String name="presetThemeChannel" value="JSPPresetThemeContainer"
            advanced="true"/>
        <String name="customThemeChannel" value="JSPCustomThemeContainer"
            advanced="true"/>
    </Properties>

</Provider>
```

The sample RouterContainerProvider channel XML fragment is in the `RCChannel.xml` file.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">

        <Container name="JSPRouterContainer"
            provider="RouterContainerProvider" advanced="true">
            <Properties>
                <String name="title" value="Router Container"/>
                <String name="description" value="Router container"/>
                <Boolean name="isEditable" value="true" advanced="true"/>
                <String name="editType" value="edit_complete"
                        advanced="true"/>
                <String name="contentPage" value="router.jsp"/>
            </Properties>

            <Available>
                <Reference value="JSPTabContainer"/>
                <Reference value="JSPTableContainer"/>
            </Available>

            <Selected>
                <Reference value="JSPTabContainer"/>
```

```
        </Selected>

        <Channels>
        </Channels>
    </Container>
```

For RouterContainerProvider, use the `psadmin add-display-profile` sub command to upload the `RCProvider.xml` file and `RCChannel.xml` file fragments in to the display profile. See the *Sun Java System Portal Server 7.1 Command Line Reference* for more information on this sub command.

4 Develop the templates or JSP files for the ContainerProvider.

The sample RouterContainerProvider requires the following JSP files:

`router.jsp`	This file includes the logic for displaying the tab or the table Desktop based on the user's selection.
	`<%-- router.jsp --%>`
	
	`<%@ page import="com.sample.providers.containers.router.RouterContainerProvider"%>`
	
	`<%`
	`RouterContainerProvider rcp = (RouterContainerProvider) pageContext.getAttribute("JSPProvider");`
	`%>`
	`<%= rcp.getContainerProviderContext().getContent(request, response, null, rcp.getSelectedChannel())%>`

`routeredit.jsp`

This file includes the logic for displaying the tab or the table Desktop based on the user's selection.

```
<%-- routeredit.jsp  --%>

<%@ page import="com.sample.providers.containers.router.
RouterContainerProvider,
com.sun.portal.providers.Provider,java.util.List,
java.util.Map,java.util.ArrayList"
%>

<%@ page session="false" %>

<%@ taglib uri="/tld/jx.tld" prefix="jx" %>
<%@ taglib uri="/tld/desktop.tld" prefix="dt" %>
<%@ taglib uri="/tld/desktopProviderContext.tld"
prefix="dtpc" %>
<%@ taglib uri="/tld/desktopContainerProviderContext.tld"
prefix="dtcpc" %>
<%@ taglib uri="/tld/desktopTheme.tld" prefix="dttheme" %>

<%
    RouterContainerProvider rcp = (RouterContainerProvider)
    pageContext.getAttribute("JSPProvider");
    String selectedChannel = rcp.getSelectedChannel();
    pageContext.setAttribute( "rcp", rcp );
%>

<dt:obtainContainer container="$rcp">
<dtpc:providerContext>

    <dtpc:getStringProperty key="fontFace1" id="fontFace"/>
    <jx:declare id="fontFace" type="java.lang.String"/>
    <%
        String fontFace1 = (String)pageContext.getAttribute
        ("fontFace1", PageContext.REQUEST_SCOPE);
    %>
<jsp:include page="header.jsp" flush="true"/>

<form action="dt" name="routeredit_form" method=POST
enctype="application/x-www-form-urlencoded">
    <input type=HIDDEN name="action" size=-1 value="process">
    <input type=HIDDEN name="provider" size=-1
    value="JSPRouterContainer">

    <dt:getAvailableChannels id="availableChannel"/>

    <CENTER>
    <table border=0 cellspacing=1 cellpadding=0 width="75%"
    align="center">
        <tr>
            <td bgcolor="#666699"><b><font face="<%=fontFace%>"
            color="#FFFFFF" size="+1">Please select a desktop for
            your front page</b></font>
            </td>
        </tr>
```

5 Make a directory for the files in *PortalServer-DataDir*/portals/*portal-ID*/desktop/*desktoptype* directory and copy the files over to the newly created directory.

For the sample RouterContainerProvider, type:

```
mkdir PortalServer-DataDir/portals/portal-ID
/desktop/desktoptype/JSPRouterContainer
cp router.jsp PortalServer-DataDir/portals/portal-ID
/desktop/desktoptype/JSPRouterContainer/
cp routeredit.jsp PortalServer-DataDir/portals/portal-ID
/desktop/desktoptype/JSPRouterContainer/
```

The RouterContainerProvider also requires `header.jsp` to display the header on router edit page and `menubar.jsp` to display the menubar on the router edit page. Copy these files from *PortalServer-DataDir*/portals/*portal-ID*/desktop/default/JSPSingleContainerProvider directory.

```
cp PortalServer-DataDir/portals/portal-ID
/desktop/default/JSPSingleContainer/header.jsp PortalServer-DataDir/
portals/portal-ID/desktop/desktoptype/
JSPRouterContainer/
cp PortalServer-DataDir/portals/portal-ID/
desktop/default/JSPSingleContainer/menubar.jsp PortalServer-DataDir/
portals/portal-ID/desktop/desktoptype/
JSPRouterContainer/
```

6 Modify the `header.jsp` file in:

- The *PortalServer-DataDir*/portals/*portal-ID*/desktop/default/JSPTabContainer directory to add the text from `tabheader.jsp` file (as shown below).

  The HTML code to add a link to the edit page of the router container provider is shown (in bold) in the `tabheader.jsp` (see below) file. Add this HTML code to the `header.jsp` file.

  PortalServer-DataDir/portals/
  portal-ID/desktop/default/JSPTabContainer/
  header.jsp

  and

  PortalServer-DataDir/portals/
  portal-ID/desktop/default/
  JSPTableContainerProvider/header.jsp

```
<%--
    Copyright 2001 Sun Microsystems, Inc.  All rights reserved.
    PROPRIETARY/CONFIDENTIAL.  Use of this product is subject
        to license terms.
--%>
```

```
<%-- tabheader.jsp  --%>

<HTML>
    <CENTER>
    <TABLE BORDER="0" CELLPADDING="03" CELLSPACING="0"
    ALIGN="right" HEIGHT="29">
    <TR>

        <td><a href="dt?action=edit&provider=JSPRouterContainer"
        onMouseOver="over('banner_home')" onMouseOut="out('banner_home')">
        <img name="banner_home" src="<dt:scontent/>/images/blueBullet.gif"
        width="13" height="9" border="0" alt=""></a></td>

        <td><a href="dt?action=edit&provider=JSPRouterContainer"
        onMouseOver="over('banner_home')" onMouseOut="out('banner_home')"
        class="noUnderline"> <span class="banner-links">Desktop Preference
        </span></a> </td>
    </TR>
    </TABLE>
    <img src="<dt:scontent/>/desktop/images/nothing.gif" height="8"
    width="0" border="0" alt=""><br>
</HTML>
```

- The *PortalServer-DataDir*/portals/*portal-ID*/desktop/default/JSPTableContainerProvider to add the text from tableheader.jsp file (as shown below).

  The HTML code to add a link to the edit page of the router container provider is shown (in bold) in the tableheader.jsp (see below) file.

```
<%--
    Copyright 2001 Sun Microsystems, Inc.  All rights reserved.
    PROPRIETARY/CONFIDENTIAL.  Use of this product is
        subject to license terms.
--%>
<%-- tableheader.jsp  --%>

<HTML>
    <CENTER>
    <TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0" WIDTH="100%">
    <TR>
        <td bgcolor="#CC0000" colspan="2"><img src="<dt:scontent/>
        /images/spacer.gif" width="1" height="3" alt=""></td>
    </TR>
    <TR>

        <td><a href="dt?action=edit&provider=JSPRouterContainer"
        onMouseOver="over('banner_home')" onMouseOut="out('banner_home')">
        <img name="banner_home" src="<dt:scontent/>/images/blueBullet.gif"
```

```
            width="13" height="9" border="0" alt=""></a></td>

            <td><a href="dt?action=edit&provider=JSPRouterContainer"
            onMouseOver="over('banner_home')" onMouseOut="out('banner_home')"
            class="noUnderline"> <span class="banner-links">Desktop Preference
            </span></a> </td>
        </TR>
        </TABLE>
        <img src="<dt:scontent/>/desktop/images/nothing.gif" height="8"
        width="0" border="0" alt=""><br>
    </HTML>
```

7 Open a browser and log in to the Portal Server administration console to change the default channel name to your ContainerProvider.

The URL to access the administration console is `http://`*hostname*`:`*port*`/psconsole`. For the sample RouterContainerProvider, change the default channel name to JSPRouterContainer.

8 Save the settings and log out of the administration console and log in as the default user into the portal Desktop.

9 Select the Desktop Preference Link from the menubar.

The sample RouterContainerProvider provides the page to select Tabbed or Table Desktop.

◆ ◆ ◆ CHAPTER 11

Overview of the Portlets

This chapter contains the following sections:

What is a Portlet?

Portlet refers to pluggable web components that process requests and generate content within the context of a portal. Conceptually, portlets are equivalent to the providers. The portlets process requests coming from the portal and generate content for inclusion on the desktop. The portlets generate dynamic content and interact with web clients through a request and response paradigm.

The portlets only generate markup fragments and not complete documents. The portal aggregates portlet markup fragments into a complete portal page. Portlets are not directly bound to a URL; they have URL rewriting functions to create hyper links within their content. Portlets have predefined portlet modes (VIEW, which is mandatory, EDIT, and HELP, which are optional) described in their deployment descriptor and window states that indicate the function that the portlet performs and the amount of real estate in the portal desktop. They can store transient data in the portlet session in two different scopes: the application-wide scope and the portlet private scope.

In the Sun Java System Portal Server software, portlets are managed by a Portlet Container. The Portlet Container manages portlets in the same way that the web container manages servlets. The portlet container manages the life cycle of the portlets within the Portal Server. The life cycle refers to the creation, operation, and destruction of the portlets. The portlet container is also responsible for routing the incoming requests to the correct portlet. The portlet container also provides a persistent storage mechanism for the storage of portlet preferences. The portlet states for Portal Server are hardcoded in the portlet container and is not extensible.

Overview of Developing and Deploying Portlets

The portlet API includes a GenericPortlet abstract class that implements the Portlet interface and the PortletConfig interface. This class provides access to the following common portlet functionality:

Dispatch processing for portlet URLs	The render method is implemented to determine current portlet mode and to call the appropriate method.
Accessor for the portlet's configuration	The class includes an easy mechanism to obtain portlet's initialization parameters.
Accessor for the portlet's context	The class includes an accessor method to obtain a reference to the PortletContext interface that enables portlets to interface with the container.

▼ To develop and deploy portlets

1. **Develop the portlet class files.**

   A portlet must implement the `javax.portlet.Portlet` interface. This interface defines the basic behavior that the portlet container expects from a portlet.

2. **Create the portlet specific XML fragments and package it into a WAR file.**

3. **Deploy the Portlet web application.**

4. **Create and view the Portlets.**

◆ ◆ ◆ CHAPTER 12

Deploying Struts Application as a Portlet in Portal Server

This chapter describes how to deploy any existing struts application as a JSR 168 portlet in Portal Server. Using the steps mentioned in this document, the entire Struts application can be deployed within a channel on the portal server desktop. This chapter contains the following sections:

Preparing the Struts Application

This section explains how to prepare the Struts application.

Introduction

The extended struts framework shipped with Portal Server is an extension of Struts version 1.2.4. This requires that you must download Standard Struts binary, version 1.2.4, from the struts archive page, and the application must be tested as a standalone application, using standard `struts.jar` file. This is to ensure that you have the proper version of all the supporting JARs required by the struts framework.

Install Portal Server 7.1 for extended struts framework (`struts.jar` file) and supporting components (`strutssupport.jar`, `portlet.jar`) required to deploy Struts application as JSR 168 portlet.

Modify Struts Application

To deploy any struts application as a portlet, the struts application is required to follow these guidelines:

1. The Struts application must abide by the restrictions applicable to any application running in the Portal Server. For example, the request parameters in the struts application can not use keywords reserved by the portal server. The list of reserved words include `action`, `provider`, `targetprovider`, `containerName`, `last`, `page"`, `error`, `container`, `selected`, `editChannelName`, `targetPortletChannel`, and `currentChannelMode`.
2. All the forms and links must be created using struts tag library. Struts tag library provide `<html:form>` and `<html:link>` for this purpose.
3. JSP and HTML must not have HTML title, body, frame and base tags. JSP must not use forward and/or redirect.

Obtain Portlet Objects in Struts Application

It is possible for struts application to get hold of portlet objects such as `ActionRequest` and `ActionResponse`. This may be required, for example, to implement EDIT functionality. However, if portlet objects are not used properly, the use of portlet objects in struts application may make it portal dependent and result in the struts application unusable as a standalone application.

The struts `Action` class can obtain `javax.portlet.ActionRequest` and `javax.portlet.ActionResponse` objects using the following calls:

```
ActionRequest aReq = (ActionRequest) request.getAttribute("javax.portlet.request");
ActionResponse aRes = (ActionResponse) request.getAttribute("javax.portlet.response");
```

The above two statements return `javax.portlet.RenderRequest` and `javax.portlet.RenderResponse` respectively, when called from a JSP page.

Session Information

If any struts application, deployed as a portlet, is invalidating the session using `session.invalidate()`, the session obtained by the struts-portlet bridge becomes the invalid one. Because of this, the bridge is unable to store rendering related information. In application server, struts application, deployed as a portlet, must not use `session.invalidate()` as the same session is used by struts portlet bridge.

Creating and Modifying XML Files

This section explains how to create and modify XML files.

Modifying `struts-config.xml` File

Change the `RequestProcessor` to `org.apache.struts.action.PortletRequestProcessor` or `org.apache.struts.tiles.PortletTilesRequestProcessor`, if the application is using Tiles.

For example:

```
<controller
contentType="text/html;charset=UTF-8"
debug="3"
locale="true"
processorClass="org.apahce.struts.action.PortletRequestProcessor">
<!-- The "input" parameter on "action" elements is the name of a
local or global "forward" rather than a module-relative path -->
<set-property property="inputForward" value="true"/>
</controller>
```

Creating `portlet.xml` File

Every portlet WAR must have one `portlet.xml` file in the `WEB-INF` directory of the web application. When creating the `portlet.xml` file, note that:

- The Portlet class must be `org.apache.struts.action.StrutsPortlet`.
- The `initPage` *init* parameter is mandatory and its value must be the welcome page of the struts application. This can be a direct reference to a JSP file (such as `/index.jsp`) or it can be a reference of Action Mapping Definition (such as `/welcome.do`).
- The *editPage init* parameter is not mandatory. If specified, portlet mode `EDIT` must also be specified in `<supports>` tag and vice-versa.
- The *helpPage init* parameter is not mandatory. If specified, portlet mode `HELP` must also be specified in `<supports>` tag and vice-versa. Note that the help page support is limited to a single page and it can not provide navigation to any other page within struts application.
- The *factoryName init* parameter is mandatory and must be set to `com.sun.portal.struts.wrapper.PSServletObjectsFactory`.
- All the init parameters associated with the `ActionServlet` as defined in `web.xml` file must also be configured as init parameter in `portlet.xml` file.

- The URL mapping used for `ActionServlet` as defined in `web.xml` file must be configured as an init parameter of the portlet.

Here is a sample `portlet.xml` file for struts-portlet application:

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/
portlet/portlet-app_1_0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://java.sun.com/xml/ns/portlet
/porlet-app_1_0.xsd" version="1.0">

<portlet>
    <portlet-name>StrutsPortlet</portlet-name>
    <portlet-class>org.apache.struts.action.
    StrutsPortlet</portlet-class>
    <init-param>
        <name>initPage</name>
        <value>/index.jsp</value>
    </init-param>
    <init-param>
        <name>helpPage</name>
        <value>/tour.htm</value>
    </init-param>
    <init-param>
        <name>editPage</name>
        <value>/edit.jsp</value>
    </init-param>
    <init-param>
        <name>factoryPage</name>
        <value>com.sun.portal.struts.wrapper.
        PSServletObjectsFactory</value>
    </init-param>
    <init-param>
        <name>config</name>
        <value>/WEB-INF/struts-config.xml,/WEB-INF/
        struts-config-registration.xml</value>
    </init-param>
    <init-param>
        <name>servletPage</name>
        <value>*.do</value>
    </init-param>
        <expiration-cache>0</expiration-cache>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>HELP</portlet-mode>
            <portlet-mode>EDIT</portlet-mode>
        </supports>
```

```
    <portlet-info>
        <title>StrutsPortlet</title>
    </portlet-info>
    </portlet>
</portlet-app>
```

Building and Deploying the Web Application as a Portlet Application

Follow the procedure to deploy the struts application as a portlet.

▼ To Deploy the Struts Application as a Portlet

1. **Replace standard** `struts.jar` **file, in the** `WEB-INF/lib` **directory, with the extended** `struts.jar` **file shipped Portal Server.**

2. **Add** `strutssupport.jar` **file and** `portlet.jar` **file shipped with Portal Server in the** `WEB-INF/lib` **directory.**

3. **Copy the newly created** `portlet.xml` **file and modified** `struts-config.xml` **file to the** `WEB-INF` **directory.**

4. **Create the** `.war` **file for the application.**

5. **Deploy the newly created WAR file using the** `psadmin deploy-portlet` **command. For example, type** **`./psadmin deploy-portlet -u`** *amadmin* **`-f`** *passwordfile* **`-p`** *portalId* **`-i`** *portalinstance* **`-g`** *warfile*.

◆ ◆ ◆ CHAPTER 13

Deploying JSF Application as a Portlet in Portal Server

This chapter describes how to deploy any existing JavaServer Faces (JSF) application as a JSR 168 portlet in Portal Server. Using the steps mentioned in this document, the entire JSF application can be displayed within a channel on the portal server desktop. This chapter contains the following sections:

Overview

This section contains the following:

Introduction

A `jsf-portlet.jar` file is included with the Portal Server to enable communication between the Portal Server and JSF. This component allows the execution of commands (to render the information, to perform actions like EDIT, HELP) received from Portal Server and pass the data (such as user defined parameters, or user input in general) to the JSF-based web application and/or to Portal Server.

State Information and High Availability

The JSF application embedded in a JSF portlet can view all user interactions with the portal page that are outside the user interface for the JSF portlet itself as if they were page reloads. The JSF

Portlet maintains whatever state information is needed by the JSF application as other pages are selected or while the user interacts with the portal in other ways. The JSF portlet leverages the portlet container HTTP session failover capabilities to enable highly available JSF applications within portlets.

Accessing Portlet APIs

Developers can access the portlet APIs from the `FacesContext` object as shown here:

```
FacesContext facesContext = FacesContext.getCurrentInstance();
PortletRequest pRequest = (PortletRequest)facesContext.
getExternalContext().getRequest();
```

Use the command to find the portlet window state (like Maximize and Normalize).

```
WindowState windowState = pRequest.getWindowState();
```

Mapping Actions of JSF Application to Portal Application and Vice-Versa

TABLE 13–1 JSF to Portal Mapping

When the JSF Application	On the Portal
Execute any action in the application	The `processAction()` method of JSF Portlet is called which calls the `execute()` method of JSF Lifecycle. Then the `render()` method of JSF Portlet is called which calls the render method of JSF Lifecycle and returns content using `PortletRequestDispatcher.include()`.
Executing the action results in a error	The `processAction()` method of JSF Portlet is called which calls the `execute()` method of JSF Lifecycle. Then `render()` method of JSF Portlet is called which calls the `render()` method of JSF Lifecycle and returns the error message using `PortletRequestDispatcher.include()`.

TABLE 13–1 JSF to Portal Mapping *(Continued)*

When the JSF Application	On the Portal
The scope of the JSF Component or backing beans is request	When a JSF application is running in a servlet environment, the JSF request begins and ends within the scope of a servlet request (a user request). However, when a JSF application is running in a portlet environment, the JSF request lifecycle is split in two portlet requests. All JSF lifecycle phases but render happen during the portlet processAction request, with the JSF lifecycle render phase happening during the portlet render request.

TABLE 13–2 Portal to JSF Mapping

On a Portal	The JSF Application
Click on a JSF portlet Edit button	Edit page is displayed if portlet init parameter *com.sun.faces.portlet.INIT_EDIT* is set to the edit page; otherwise, a message indicating what needs to be done is displayed.
Click on a JSF portlet Help button	Help page is displayed if portlet init parameter *com.sun.faces.portlet.INIT_HELP* is set to the help page; otherwise, a message indicating what needs to be done is displayed.
Click on a JSF portlet Maximize button	The *render()* method of JSF Portlet is called which calls the render method of JSF Lifecycle and it returns content using `PortletRequestDispatcher.include()`. Therefore, the maximize window displays the same context which were shown on portlet window before clicking maximize button.
Click on a JSF portlet Minimize button	Request is handled at the portlet level and the JSF application remains unaware of it. The desktop displays the minimized window as it displays for any other JSR 168 portlet.
Click on a Normalize button of the minimized JSF portlet	The `render()` method of JSF Portlet is called which calls the render method of JSF Lifecycle and it returns content using `PortletRequestDispatcher.include()`. Therefore, the normalized window displays the same content which was shown on portlet window before clicking the minimize button.

TABLE 13–2 Portal to JSF Mapping *(Continued)*

On a Portal	The JSF Application
Click on a JSF portlet Detach button	The `render()` method of JSF Portlet is called which calls the render method of JSF Lifecycle and it returns content using `PortletRequestDispatcher.include()`. Portal server uses this content to display in a new window. Therefore, the detached window displays the same content which was shown on portlet window before clicking the detach button.
After detaching, click on a JSF portlet Attach button	The `render()` method of JSF Portlet is called which calls the render method of JSF Lifecycle and it returns content using `PortletRequestDispatcher.include()`. Portal server uses this content to display in the portlet window. The content shown is same as was shown in the detached window before clicking the attach button.
Remove the JSF portlet from a page and then add it again	On removal, the request is handled at the portal/portlet level and on adding it again, the `render()` method is called. The jsf-portlet window displays the same content which was shown on portlet window before removing this portlet (that is, the state is maintained). The remove and add have to occur in the same login session while the same `DesktopContext` object exists. For example, if the desktop session reap interval setting is set low enough (say 30 second), and you remove a JSF portlet, then wait two minutes and then add it again, the state will be lost.
Click reload of the portal page	The `render()` method of JSF Portlet is called which calls the render method of JSF Lifecycle and it returns content using `PortletRequestDispatcher.include()`. The jsf-portlet window displays the same content which was shown on portlet window before clicking the reload button.
Click on another tab and then click back on the tab that contains the JSF portlet	The `render()` method of JSF Portlet is called which calls the render method of JSF Lifecycle and it returns content using `PortletRequestDispatcher.include()`. The jsf-portlet window displays the same content which was shown on portlet window before clicking on other tab.

TABLE 13–2 Portal to JSF Mapping *(Continued)*

On a Portal	The JSF Application
Click on the Finish button of the edit page of some other channel, thereby causing a refresh of the portal page containing the JSF portlet.	The `render()` method of JSF Portlet is called which calls the render method of JSF Lifecycle and it returns content using `PortletRequestDispatcher.include()`. Therefore, the jsf-portlet window displays the same content which was shown on portlet window before clicking the edit button.
Execute any action on some other channel, thereby causing the portal page containing JSF portlet to be refreshed	The `render()` method of JSF Portlet is called which calls the render method of JSF Lifecycle and it returns content using `PortletRequestDispatcher.include()`. The jsf-portlet window displays the same content which was shown on portlet window before executing any action on some other channel.

Converting JSF-based Applications to JSF Aware Portlets in Portal Server

Follow the procedure to convert JSF-based application to portlets.

▼ To Convert JSF-based Applications to Portlets

1. **Copy `jsf-portlet.jar` from *PortalServer7–base*`/lib` directory to `WEB-INF/lib` directory of the application.**
2. **Add a new deployment descriptor for the portlet by creating a `portlet.xml` file.**

   The `portlet.xml` file must placed in the `WEB-INF` directory of the application.
3. **In the `portlet.xml` file, set the portlet parameter *com.sun.faces.portlet.INIT_VIEW* to point to the first page of your portlet.**
4. **Modify the JSP pages as follows:**

   a. **Remove the `<html>`, `<head>`, and `<body>` tags.**

   b. **Modify use of forward and redirect as the new page will replace the existing portal pages.**

   c. **Remove all the HTML tags and JavaScript calls which are not allowed (as per JSR 168 specification).**

5. **(Optional) Set the portlet parameter** `com.sun.faces.portlet.INIT_EDIT` **to point to the edit page of your portlet in the** `portlet.xml` **file to provide EDIT functionality for the JSF portlet.**

6. **(Optional) Set the portlet parameter** `com.sun.faces.portlet.INIT_HELP` **to point to the help page of your portlet in the** `portlet.xml` **file to provide HTLP functionality for the JSF portlet.**

7. **Deploy the WAR file using the** `psadmin deploy-portlet` **command. For example, type** **`psadmin deploy-portlet -u`** *amadmin* **`-f`** *passwordfile* **`-v -d`** *dn* **`-p`** *portalID* **`-i`** *instanceID warfile*.

8. **Create a new portlet channel and add it to the desired container.**

◆ ◆ ◆ CHAPTER 14

Extending the GenericPortlet Abstract Class

This chapter contains the following sections:

Introduction to Extending the GenericPortlet Abstract Class

This chapter includes instructions for extending the GenericPortlet abstract class. It includes instructions to create a sample portlet named PrefPortlet. In the view mode, this portlet displays a salutation. In the edit mode of PrefPortlet, the user can change that salutation. The modified salutation is saved into the preference to be used for subsequent requests to the portlet.

Developing the Class File

This section includes:

GenericPortlet Class

This section includes:

Methods to Override

When extending from the GenericPortlet abstract class, the following methods should be overridden:

`public void init(PortletConfig config)`
The container calls this method once when the portlet object is created.

`protected void doView(RenderRequest req, RenderResponse resp)`
The render method calls this method when the portlet is in VIEW mode. The GenericPortlet class' default implementation only throws an exception.

`protected void doEdit(RenderRequest req, RenderResponse resp)`
The render method calls this method when the portlet is in EDIT mode. The GenericPortlet class' default implementation only throws an exception.

`protected void doHelp(RenderRequest req, RenderResponse resp)`
The render method calls this method when the portlet is in HELP mode. The GenericPortlet class' default implementation only throws an exception.

`public void processAction(ActionRequest req, ActionResponse resp)`
The GenericPortlet class includes a default implementation that only throws an exception.

`public void destroy()`
The container calls this method on the portlet when the portlet is being taken out of service.

Render Request Processing

The request from the client is encapsulated in a RenderRequest object. The portlet uses this object to access information about the request that was received from the client by the Portal Server. Typically, the current portlet mode and request parameters included in the incoming request is the information that is accessed. The portlet's response is encapsulated in a RenderResponse object. Portlet use this object to set the content type of the response and send their portlet content to the portlet container.

Each request object is valid only within the scope of a particular `processAction()` or `render()` method call.

Action Request Processing

Action URL requests from the client are encapsulated in an ActionRequest object. The portlet uses ActionRequest objects to gain access to the details of the incoming HTTP request from the client. The portlet's response to the action request is encapsulated in the ActionResponse object. The portlet uses this object to set the new portlet mode (typically the VIEW mode) and send information to the subsequent render request after action request has been proceessed.

Portlet Preferences

The portlet specification mandates that portlet container provide persistent store for portlet preferences. Portlets can only access their preferences during requests (render or action). Portlet preferences are encapsulated by the PortletPreferences interface.

Portlets can update their preferences within the processing of the `processAction()` method. An exception is thrown if the portlet attempts to update preferences during a render request.

User preferences are stored and accessed through a `PortletPreferences` object. A handle to this object can be obtained by using both `RenderRequest.getPreferences()` and `ActionRequest.getPreferences()`, depending from which method the attempt to access the preferences is made. An individual preferences is called using `getValue()` or `getValues()`, for single string value or multiple string values respectively. The methods `setValue()` and `setValues()` modifies the preference, but no changes are stored in the datastore until the method `store()` is called. All user preferences must be defined in the portlet deployment descriptor.

Preferences defined at the organization will affect the user's preference only if the user's preference has not been defined and only until the user modifies the individual preference. Once preferences have been modified, it will retain its value, even if blank, until the `reset()` method is called on the preference.

LDAP attributes can only be accessed through the portlet implementation of a user info Map. All attributes needed within the portlet must be defined in the portlet deployment descriptor. The attribute map of all such defined attributes can be retrieved using either `ActionRequest.getAttribute(PortletRequest.USER_INFO)` or `RenderRequest.getAttribute(PortletRequest.USER_INFO)`. Each attribute can then be retrieved using `Map.get("Attribute Name")`.

Only user attributes defined in the deployment descriptor can be read. The user info map returned is unmodifiable.

Example Portlet Class File

Following example contains the class file for the sample PrefPortlet.

EXAMPLE 14–1 `PrefPortlet.java` File

```
package examples;

import javax.portlet.GenericPortlet;
import javax.portlet.ActionRequest;
import javax.portlet.RenderRequest;
import javax.portlet.ActionResponse;
import javax.portlet.RenderResponse;
```

EXAMPLE 14–1 PrefPortlet.java File *(Continued)*

```
import javax.portlet.PortletException;
import javax.portlet.PortletURL;
import javax.portlet.PortletMode;
import javax.portlet.PortletPreferences;
import javax.portlet.WindowState;
import java.io.IOException;
import java.io.PrintWriter;

public class PrefPortlet extends GenericPortlet {
    public void processAction(ActionRequest request,
        ActionResponse response)
        throws PortletException {
        // process the salutation set by the user
            in the edit mode.
        String salutation = request.getParameter("SALUTATION");
        try {
            PortletPreferences pref = request.getPreferences();
            pref.setValue("salutation", salutation);
            pref.store();
        } catch (Exception e) {
            throw new PortletException(e.getMessage());
        }
        // return the user back to the view mode and normal state
        response.setPortletMode(PortletMode.VIEW);
        response.setWindowState(WindowState.NORMAL);
    }

    public void doView(RenderRequest request,RenderResponse response)
        throws PortletException,IOException {
        // displays the salutation stored in the preference.
        PortletPreferences pref = request.getPreferences();
        String salutation = pref.getValue("salutation","");
        response.setContentType(request.getResponseContentType());
        PrintWriter writer = response.getWriter();
        writer.write("Hello " + salutation);
    }

    public void doEdit(RenderRequest request,RenderResponse response)
        throws PortletException,IOException {
        PortletURL actionURL = response.createActionURL();
        response.setContentType(request.getResponseContentType());
        PrintWriter writer = response.getWriter();
        writer.print("<form method=\"post\" action=\""
            + actionURL.toString());
        writer.println("\">");
            writer.println("<center><p>Salutation: <input type=
```

EXAMPLE 14–1 PrefPortlet.java File *(Continued)*

```
            \"text\" name=\"salutation\"></p>");
            writer.println("<input type=\"submit\" value=\"Submit\">");
            writer.println("</form>");
    }

    public void doHelp(RenderRequest request, RenderResponse response)
        throws PortletException {
        response.setContentType(request.getResponseContentType());
        try {
            response.setContentType(request.getResponseContentType());
            PrintWriter writer = response.getWriter();
            writer.write("Pref Portlet Help<p><p>");
        } catch (IOException e) {
            throw new PortletException("PrefPortlet.doHelp exception", e);
        }
    }
}
```

Compiling the Portlet

The portlet API is contained within a single JAR file. When compiling a portlet, include the classpath to the JAR file in order for the compilation to succeed. Ensure that *PortalServer-base*`/lib/portlet.jar` is in your classpath when compiling the portlet. For example, to compile, type:

```
javac -classpath PortalServer-base/lib/portlet.jar PrefPortlet.java
```

Note – This compiled class file must be included in the WAR file that will be deployed on the Portal Server using the `psadmin deploy` sub command.

Creating a Portlet Web Application

Assemble the portlet class file and the XML fragments into a portlet web application. A portlet web application must be packaged as a web application archive (WAR) file.

The portlets contained within the WAR file are described using the deployment descriptor. The portlet deployment descriptor must be in the `portlet.xml` file. The structure of the `portlet.xml` file is defined by the `portlet.xsd` file located in the *PortalServer-base*`/dtd` directory. The `portlet.xml` file must be placed into the same location as the `web.xml` file.

Note – The Portal Server software includes its own portlet deployment descriptor in the `sun-portlet.xml` file. This file includes information about how the portlet container within the Portal Server software should manage the deployed portlets.

▼ To create a portlet web application

1 **Create a complete `portlet.xml` file that includes the declaration for all the portlets.**

"Creating a Portlet Web Application" on page 189 contains the `portlet.xml` file that includes the declaration for the sample portlet. The `portlet.xml` file includes name, class, and cache information. Since the sample portlet also uses preferences, the preference setting is also included in the `portlet.xml` file. Also PrefPortlet supports the EDIT as well as HELP mode; so the supported modes are specified in the `portlet.xml` file.

`portlet.xml` File for the PrefPortlet

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet
/portlet-app_1_0.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:portlet="http://java.sun.com/xml/ns/portlet"
xsi:schemaLocation="http://java.sun.com/xml/ns/portlet
/portlet-app_1_0.xsd /opt/SUNWps/dtd/portlet.xsd" version="1.0">
    <portlet>
        <portlet-name>PrefPortlet</portlet-name>
        <portlet-class>examples.PrefPortlet</portlet-class>
        <expiration-cache>0</expiration-cache>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>EDIT</portlet-mode>
            <portlet-mode>HELP</portlet-mode>
        </supports>
        <portlet-info>
            <title>PrefPortlet</title>
            <keywords>Hello, world, test</keywords>
        </portlet-info>
        <portlet-preferences>
            <preference>
                <name>name</name>
                <value>World</value>
            </preference>
        </portlet-preferences>
    </portlet>
</portlet-app>
```

2. **Create the `web.xml` file. A web application also requires a `web.xml` file.**

   If you have servlets for your portlet web application, then the servlet definition can be incorporated in the `web.xml` file.

   `web.xml` File for PrefPortlet

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//
DTD Web Application 2.3//EN" "http://java.sun.com/dtd/
web-app_2_3.dtd">
<web-app>
    <display-name>Portlet Examples</display-name>
</web-app>
```

3. **Package everything into a WAR file.**

   For the sample portlets, create an examples directory where all the files of the WAR will be placed. Then within the examples directory, create the `WEB-INF` directory. All the portlet files will be placed into the `WEB-INF` directory. For example, your directory hierarchy may look like this:

```
examples/WEB-INF/
        /web.xml
        /portlet.xml
        /classes/examples/PrefPortlet.class
```

4. **Create the WAR file. That is, change directories to `examples` and type the following command:**

```
jar -cvf examples.war *
```

Deploying the Application

The Portal Server software includes a deployment tool which will take your WAR file and deploy it into the portal server. Before deploying the WAR file, determine where to place the portlets inside the organization.

With the following command, the sample portlets can be deployed into the organization.

```
psadmin deploy-portlet -u uid -f passwordfile
 -g|-d distinguishedName -p portal1 examples.war
```

Note that the Portal Server deploys portlet applications as separate web applications within the web container. Portlet preferences are stored in the display profile with the portlet channel definition. The display profile is updated with a new provider for each portlet contained within the portlet web application. For more information on the `psadmin deploy-portlet` sub command, see Command Line Reference.

Creating Channels from the Deployed Portlets

Once the portlet is deployed, the portal server is aware of the portlet defined in the application. You can start to create channels based on the portlet. To create channels, see the Portal Server administration console online help for more information.

Debugging the Portlet

A logging mechanism is provided in the PortletContext, accessible from `GenericPortlet.getPortletContext()` method. The `PortletContext.log()` method will print a time-stamped message to the web container log and also to the portal log. There are two places to find the logs for the portlet:

WebServer-base`/Instance/logs/errors`
Includes `PortletContext.log()` messages

PortalServer-base`/portals/Portal-ID/logs/Instance/portal.0.0.log`
Includes `PortletContext.log()` messages. These messages are logged at the log level INFO.

◆ ◆ ◆ CHAPTER 15

Using Inter Portlet Communications

This chapter explains includes the following sections:

Developing Inter Portlet Communication Portlets

This section explains about the steps to develop Inter Portlet Communication (IPC) sample portlets. The topics include:

- Inter Portlet Communication API
- Event Generation and Subscription
- Event Handling Life Cycle
- Infinite Event Cycle Detection
- Deterministic Behavior
- Failure and Exception Handling

Inter Portlet Communication API

Portlets can communicate with each other even if they are in different web applications. It is assumed that all the sample portlets are on the same instance of a Portal Server. The Inter Portlet Communication API uses event generation and notification to convey the information or data among portlets. The event notification occurs for the portlets, which are registered themselves for listening to that particular event.

The IPC API is located in the `com.sun.portal.portlet` package. Portlets that are interested in receiving an event implements a single interface, `PortletEventListener`.

```
public interface PortletEventListener
{
```

```
    public void handleEvent(EventRequest ereq, EventResponse eres)
 throws PortletException;
}
```

You can use `EventRequest` to obtain the event name and event payload data. You can obtain Event payload data either by getting the event stream and reading from it or by calling `getEventData()` method which returns an Object and then casting it appropriately. If `getEventData()` is called after getting the event stream an `IllegalStateException` is thrown. Similarly, if `getEventStream()` is called after `getEventData()` an `IllegalStateException` is thrown.

```
public interface EventRequest()
    {
        public String getEventName();
        public InputStream getEventStream();
        public Object getEventData();
    }
```

You can use `EventResponse` to set the render parameters, so that the information can be passed on to the render method after processing the event received in the `handleEvent()` method.

```
public interface EventResponse()
    {
        public void setRenderParameters(Map parmMap);
        public void setRenderParameter(String key, String value);
        public void setRenderParameter(String key, String[] values);
    }
```

The portlets can generate events only from within the `handleEvent()` or `processAction()` methods. Event can be generated by instantiating `PortletEventBroker` and calling `createEvent()` method on it. The `PortletEventBroker` constructor throws an `IllegalStateException` if you call from methods other than the `handleEvent()` or `processAction()`.

```
public class PortletEventBroker
{
    //Make sure that the call is coming from handleEvent()
    or from processAction()
    public PortletEventBroker(PortletRequest pr);

    //to create an event ...
    public PortletEvent createEvent(String eventName)
    throws EventNotRegisteredException;
}
```

You can then use the `PortletEventBroker` instance to create an event by calling the `createEvent()` method.

```
public interface PortletEvent
    {
        public String getEventName();
        public OutputStream getEventStream();
        public void setEventData(Serializable s);
        public void fire();
    }
```

You can set the event data can on the event stream which can be obtained by calling `getEventStream()` method. Alternatively, event data can be set by calling `setEventData()` method. If after obtaining the event stream, attempt to call `setEventData()` throws the `IllegalStateException` exception. Similarly, after calling `setEventData()` an attempt to call the `getEventStream()` throws the `IllegalStateException` exception. The event is then fired by calling the `fire()` method.

Event Generation and Subscription

All the portlets which are interested in listening or generating an event must declare it in the `sun-portlet.xml` file as shown below:

```
<portlet>
    <portlet-name>NAME</portlet-name>
    .
    <!-- Other declarations -->
    .
    <events>
        <generates-event>NAME</generates-event>
        <generates-event>NAME</generates-event>
            .
            .
            .
        <consumes-event>NAME</consumes-event>
        <consumes-event>NAME</consumes-event>
            .
            .
            .
    </events>
</portlet>
```

If a portlet requests an event, which it has not declared in the `sun-portal.xml` file, an exception `NotRegisteredException` is thrown. Wildcards cannot be used for declaring the events that are generated. Portlets interested in consuming all the events can use wildcard character (*) as shown below:

```
<portlet>
    <portlet-name>NAME</portlet-name>
    .
```

```
    <!-- Other declarations -->
    .
    <events>
        <generates-event>NAME</generates-event>
        <generates-event>NAME</generates-event>
            .
            .
            .
        <consumes-event>*</consumes-event>
    </events>
</portlet>
```

Event Handling Life Cycle

The event cycle starts with the response to user interaction from the inside `processAction` method. These are Generation 1 events. These events are placed in the event queue by the Portlet Container and dispatched in the order they are created. The dispatching of the events continue, until all the events in the event queue are dispatched to appropriate portlets.

Dispatching of the events amount to calling the `handleEvent` methods of the appropriate portlets. Portlets can generate events in the `handleEvent` method which are Generation next events. If a portlet has subscribed to events which are generated in different generations, it will receive the events in proper order. It means that the `handleEvent` will be called with Generation ith event first and upon completion of that method, `handleEvent` will be called with event from Generation i+1.

Infinite Event Cycle Detection

Events are generated in response to the user interaction (Generation 1) or in response to other events (Generation next). This could lead to create more generations. To control the number of generations, the `maxEventGenerations` parameter in the `desktopConfig.properties` file can be configured for maximum number of generations of events per request. When the event creation exceeds the specified maximum number, a failure event, `eventHandlingFailed` will be sent to all the participating portlets.

Deterministic Behavior

If a portlet generates events X and Y in that order, then events X and Y are delivered to the portlets in that same order. If portlet A and B are interested in Event X, either A or B can get event X first. If portlets A and B are interested in Event X, and upon receipt of that event, generate events Y and Z respectively. If portlet C is interested in Event Y and Z, then portlet C can receive events Y and Z in any order.

Failure and Exception Handling

In case of failure, `handleEvent()` method may throw `PortletException`. Container will catch that exception and stop sending the events from the event queue. Container then sends another event called `eventHandlingFailed` to all the portlets participating in that particular interaction. Container does not take any action if the `PortletException` is thrown while processing `eventHandlingFailed` event. Portlets can not generate and send any events while handling the `event eventHandlingFailed`. The portlet developer has the responsibility to take appropriate action needed.

Inter Portlet Communication Sample Portlets

This section provides the location details, Installation information, and code examples of IPC sample portlets.

Location of IPC Sample Portlets

The IPC sample portlets are located at `/opt/SUNWportal/samples/ipc`. The portlet samples in `ipc` directory are divided into `ipc1` and `ipc2` sub-directories. This is to demonstrate eventing between portlets belonging to different web application and those belonging to the same web application. The interaction between the IPC Portlet samples is as follows:

- `searchportlet` (`ipc2`) —> `listportlet` (`ipc1`)
- `priceportlet` (`ipc1`) —> `considerationportlet` (`ipc1`)
- `priceportlet` (`ipc1`) —> `decisionportlet` (`ipc2`)

Source code for the following samples is located at the `/opt/SUNWportal/samples/ipc/ipc1/src` directory.

- `considerationportlet`
- `listportlet`

Source code for the following samples is located at the `/opt/SUNWportal/samples/ipc/ipc2/src` directory.

- `decisionportlet`
- `priceportlet`
- `searchportlet`

JSP files for the portlets are located at:

- `/opt/SUNWportal/samples/ipc/ipc1/<portletname>`
- `/opt/SUNWportal/samples/ipc/ipc2/<portletname>`

Web container deployment description files are located at:

- `/opt/SUNWportal/samples/ipc/ipc1/WEB-INF`
- `/opt/SUNWportal/samples/ipc/ipc2/WEB-INF`

Installing IPC Sample Portlets

The information on overview of portlets and how to install them are provided in Chapter 11.

Code Examples

This section provides code examples for:

- Generator Portlet (ipc2)
- Listener Portlet (ipc2)
- `sun-portlet.xml` of `ipc2.war`
- `sun-portlet.xml` of `ipc1.war`

Code Example for Generator Portlet (ipc2)

```
package com.sun.portal.portlet.ipc2;

    import com.sun.portal.portletappengine.ipc.EventRequest;
    import com.sun.portal.portletappengine.ipc.EventResponse;
    import com.sun.portal.portletappengine.ipc.PortletEvent;
    import com.sun.portal.portletappengine.ipc.PortletEventBroker;
    import com.sun.portal.portletappengine.ipc.PortletEventListener;
    ...

    /**
    * The PricePortlet participates in the InterPortletCommunication.
    * This generates the event that is consumed by two portlets
    * (a)ConsiderationPortlet which is in a different web application
    * (b)DecisionPortlet which is in the same web application
    * This also consumes the event generated by the DecisionPortlet.
    */
        public class PricePortlet extends GenericPortlet implements
        PortletEventListener {
            ...
            public void processAction(ActionRequest request,
            ActionResponse actionResponse)
                throws PortletException, java.io.IOException {
                PortletEventBroker peb = new PortletEventBroker(request);
                try {
                    PortletEvent pe = peb.createEvent("priceChanged");
                    String price = null;
```

```
                Object obj = request.getParameter("price");
                if (obj != null) {
                    price = (String) obj;
                } else {
                    price = "9999";
                }
                pe.setEventData(price);
                pe.fire();

                PortletEvent pe2 = peb.createEvent("priceChanged2");
                pe2.setEventData("700");
                pe2.fire();

                PortletEvent pe3 = peb.createEvent("priceChanged3");
                pe3.setEventData("800");
                pe3.fire();

            } catch (Exception e) {
                    // Do nothing!
                }
                   actionResponse.setRenderParameters(request.getParameterMap());
            }
            ...
        }
```

Code example for Listener Portlet (ipc1)

```
package com.sun.portal.portlet.ipc1;

import com.sun.portal.portletappengine.ipc.EventRequest;
import com.sun.portal.portletappengine.ipc.EventResponse;
import com.sun.portal.portletappengine.ipc.PortletEventListener;
...

/**
* The ConsiderationPortlet participates in the InterPortletCommunication.
* This consumes the event generated by the PricePortlet which is in a
* different web application.
*/
public class ConsiderationPortlet extends GenericPortlet{
   implements PortletEventListener ...
   public void handleEvent(EventRequest ereq, EventResponse eres) {
      String data = (String) ereq.getEventData();
      eres.setRenderParameter("price", data);
   }
   ...
      }
```

Code Example for Listener Portlet (ipc2)

```
package com.sun.portal.portlet.ipc2;

import com.sun.portal.portletappengine.ipc.EventNotRegisteredException;
import com.sun.portal.portletappengine.ipc.EventRequest;
import com.sun.portal.portletappengine.ipc.EventResponse;
import com.sun.portal.portletappengine.ipc.PortletEvent;
import com.sun.portal.portletappengine.ipc.PortletEventBroker;
import com.sun.portal.portletappengine.ipc.PortletEventListener;
...

/**
* The DecisionPortlet participates in the InterPortletCommunication.
* This consumes the event generated by the PricePortlet which is in the
* same web application.
* This also generates the event that is consumed by the PricePortlet.
*/
public class DecisionPortlet extends GenericPortlet{
   implements PortletEventListener ...
   public void handleEvent(EventRequest ereq, EventResponse eres) {
       String name = ereq.getEventName();
       if (name.equalsIgnoreCase("priceChanged")) {
           String data = (String) ereq.getEventData();
           int price_data = Integer.parseInt(data);
           PortletEventBroker peb = new PortletEventBroker(ereq);
           if (price_data > 500) {
               try {
                   PortletEvent pe = peb.createEvent("highPriceEvent");
                   pe.setEventData(data);
                   pe.fire();
               } catch (EventNotRegisteredException e) {
                   e.printStackTrace();
               }
           } else {
               try {
                   PortletEvent pe = peb.createEvent("lowPriceEvent");
                   pe.setEventData(data);
                   pe.fire();
               } catch (EventNotRegisteredException e) {
                   e.printStackTrace();
               }
           }
           eres.setRenderParameter("price", data);
       } else if (name.equalsIgnoreCase("priceChanged2")) {
           eres.setRenderParameter("price2", name);
       } else if (name.equalsIgnoreCase("priceChanged3")) {
           eres.setRenderParameter("price3", name);
```

```
        }
    }
    ...
}
```

Code Example for `sun-portlet.xml` of `ipc2.war`

```
<portlet>
    <portlet-name>decisionportlet</portlet-name>
    <events>
        <generates-event>decisionMade</generates-event>
        <generates-event>highPriceEvent</generates-event>
        <generates-event>lowPriceEvent</generates-event>
        <consumes-event>priceChanged</consumes-event>
        <consumes-event>priceChanged</consumes-event>
        <consumes-event>priceChanged</consumes-event>
    </events>
</portlet>

<portlet>
    <portlet-name>priceportlet</portlet-name>
    <events>
        <generates-event>priceChanged</generates-event>
        <generates-event>priceChanged</generates-event>
        <generates-event>priceChanged</generates-event>
        <consumes-event>highPriceEvent</consumes-event>
        <consumes-event>lowPriceEvent</consumes-event>
    </events>
</portlet>
```

Code Example for `sun-portlet.xml` of `ipc1.war`

```
<portlet>
    <portlet-name>considerationportlet</portlet-name>
    <events>
        <consumes-event>priceChanged</consumes-event>
    </events>
</portlet>
```

◆ ◆ ◆ CHAPTER 16

Converting Providers to Portlets

This chapter contains the following sections:

Introduction to Converting Providers to Portlets

There are numerous differences in the way portlets and providers generate content for the desktop. In order to perform a successful conversion, the user must first understand the portlet request handling sequence. The ability to access user preferences and LDAP attributes from the portlet is vital to generating the correct content.

Converting JSPs and tag libraries is the next step. JSPProvider-based JSPs and taglibraries need extensive reworking to function within the portlet framework. There is no need for a tag to wrap all the content; so all references to `<dt:obtainChannel channel="$JSPProvider">` can be removed. All taglibraries specific to the desktop (`desktop.tld`, `desktopProviderContext.tld`) should be removed from the JSP and replaced with portlet taglibraries. Unfortunately, very few of the desktop taglibraries have portlet counterparts. The generic portlet taglibrary (`portlet.tld` in `http://java.sun.com/portlet`) has a number of useful tags. There are no handles to the portlet object available to the JSP, unlike providers. Any objects that need to be shared between external JSP and/or servlets and the portlet should be stored in the session at the application scope.

Note that there is no functionality present in the GenericPortlet abstract class to convert a template-based provider, other than the ability to use HTML files for each portlet mode.

Packaging, deploying and testing the new portlet are the last steps in this process. Portlets are deployed in a custom web-app. This means all JAR files, JSPs, templates, servlets, and properties files that the portlet uses must either be included or defined in the portlet deployment descriptor file (`portlet.xml`) or the web application deployment descriptor file (`web.xml`). A

WAR file is created that contains all the files and information needed to deploy the web-app. The `psadmin deploy-portlet` command can be used to deploy the WAR file to the web container.

Mapping ProviderAdapter to GenericPortlet

The following table can be used as a guideline for comparing the Provider API to the Portlet API, when developing or trying to convert providers to portlets.

TABLE 16–1 Mapping ProviderAdapter to GenericPortlet

ProviderAdapter	GenericPortlet
`public void init(String n, HttpServletRequest req)`	`public void init (PortletConfig config)`
`public URL processEdit (HttpServletRequest request,HttpServletResponse response)`	`public void processAction (ActionRequest request,ActionResponse response)`
	`public void render (RenderRequest request, RenderResponse resposne)`
`public StringBuffer getContent (HttpServletRequest request, HttpServletResponse response)`	`protected void doView (RenderRequest request,RenderResponse response)`
`public StringBuffer getEdit (HttpServletRequest request, HttpServletResponse response)`	`protected void doEdit (RenderRequest request,RenderResponse response)`
`public URL getHelp(HttpServletRequest req, String key)`	`protected void doHelp (RenderRequest request, RenderResponse response)`
`public String getName()`	`public String getPortletName ()`
`public String getTitle()`	`protected String getTitle(RenderRequest request)`
`public ResourceBundle getResourceBundle(String base)`	`public java.util.ResourceBundle getResourceBundle (java.util.Locale locale)`
`public ResourceBundle getResourceBundle()`	
	`public PortletConfig getPortletConfig ()`
	`public PortletContext getPortletContext ()`
	`public String getInitParameter(java.lang.String name)`

TABLE 16–1 Mapping ProviderAdapter to GenericPortlet *(Continued)*

ProviderAdapter	GenericPortlet
	`public java.util.Enumeration getInitParameterNames()`
	`public void destroy ()`
`public int getEditType()`	
`public int getWidth()`	
`public boolean isEditable()`	
`public boolean isPresentable()`	
`public ProviderContext getProviderContext()`	

The Javadocs for the Provider API can be accessed using the following URL:

`http://`*portal-host*`:`*port*`/`*portal-ID*`/javadocs/index.html`

For further reading on the Portlet API, see
`http://portals.apache.org/pluto/multiproject/portlet-api/apidocs/index.html`.

Sample Code Fragments for Provider to Portlet Conversion

This section includes some sample code fragments for Provider to Portlet conversion.

Provider to Portlet Mapping

The	Maps To (or can use)
`ProviderContext.getStringAttribute ("firstname");`	`Map ui = (Map)renderRequest.getAttribute ("javax.portlet.userinfo"); String firstname = (String)ui.get("firstname");`
`ProviderContext.getSessionProperty() /setSessionProperty()`	`PortletSession.getAttribute(), PortletSession.setAttribute()`
`ProviderContext.getDesktopURL()`	`PortletURL.RenderResponse.createRenderURL(), RenderResponse.createActionURL()`

Dispatching to a JSP

```
PortletRequestDispatcher dispatcher = pContext.getRequestDispatcher("test.jsp");
dispatcher.include(request, response);
```

Help Documentation

Help documentation for portlets follows a different scheme than in the providers. The help file name is stored as a preference and content is generated from that file the same way as any other content type.

`getHelp()` In Provider, this method returns a `helpURL` which can be pointing to a static help file.

`doHelp()` For portlets, this method has to be implemented for writing out the help content dynamically.

Title Mapping

`getTitle()` For ProviderAdapter, this method returns the title from the Display Profile `title` property.

For portlets, the title is returned by the portlet implementation.

`editType` Property

For a provider, the `editType` display profile property can be specified to determine whether the provider implementation will draw the complete edit page or a subset of it.

For portlets, the `editType` is always `EDIT_COMPLETE` and the complete page including the form, the Finish, and the Cancel buttons have to be generated by the portlet. Only the banner and footer are drawn by the `PortletEdit.jsp` in *PortalServer-DataDir*`/portals/`*portal-ID*`/desktop/default` directory.

PortletPreferences Mapping

The `ProfileProviderAdapter.getStringProperty(key)` maps to `PortletPreferences.getValue(key, default)`. The `ProfileProviderAdapter.getListProperty(key)` maps to `PortletPreferences.getValues(key, default[])`.

◆ ◆ ◆ CHAPTER 17

WSRP: Validating Registration Data

This chapter contains the following sections:

Overview of WSRP Communication

A WSRP consumer, after learning of a producer existence, attempts to connect to the producer. When the consumer initially contacts the producer, the producer describes it's capabilities and expectations through it's service description. The producer can also require a consumer to provide information about itself in order to be able to access the services offered by the producer. The producer, in its service description, defines registration property keys (and descriptions thereof). The producer expects registering consumers to provide values for these keys.

The consumer describes its capabilities to the producer and also answers questions asked by the producer by providing registration property values in its registration data. At registration time, the consumer passes the registration data to the producer.

Registering with the Producer

For producers that require registration, the consumer must register to use the producer. The consumer passes registration data to the producer and the producer validates the registration data provided by the consumer. If the registration data is deemed valid, the producer returns the consumer a registration handle. The consumer uses the registration handle for all further communications with the producer.

If in-band registration is supported in the producer, the consumer can register through the WSRP registration port type. If in-band registration is not supported by the producer, the consumer administrator must manually obtain the registration handle from the producer administrator.

Validating the Registration Data

A registration validation process can be implemented for informing the producer whether a registration (data from a consumer) is valid. A public service provider interface (the RegistrationValidator interface), povided with the Portal Server software, allows the logic that determines the validity of registration data.

The RegistrationValidator Interface

RegistrationValidator is the Java interface for the registration validation service provider interface. Where non-standard logic is required for validating registration data from consumers, implement this Java interface to validate the data with arbitrary logic.

Each producer instance can be configured with a unique registration validator. The producer uses the configured registration validator when:

- A consumer performs an inband registration
- The producer administrator adds a consumer registration out of band

The RegistrationValidator interface will be passed the registration data and service description. It uses a subset of the information contained in those obejcts to determine the return value of the `validate()` method. The return value is a code indicating success or failure. Any non-negative value indicates that the validation was successful. Any negative value indicates a validation failure.

The integer code return value can be interpreted by the producer to provide meaningful error messages to users. That is, for each RegistrationValidator implementation, resource messages must be inserted into the producer's resource bundle. When an attempt to add a registration via the administration console takes place, the administration console gets the RegistrationValidator and calls the `validate()` method. A return value greater or equal to zero indicates success (validation succeeded). A negative return value indicates that validation failed.

The administration console displays the localized human-readable message regarding the validation. Typically, this is used for error messages; but it can also be used to provide warning messages or differing success messages. Registration validation messages should be formatted as such:

<RV class name><code>=<mesage>

where:

- *RV class name* is the RegistrationValidator implementation class name
- *code* is the return code from the `validate()` method
- *message* is the human-readable error message

For example, to provide a message for RegistrationValidator class `com.sun.portal.foo.FooValidator` and for validator code `-23`:

```
com.sun.portal.foo.bar.FooValidator-23=Unknown company name
```

Note – Javadocs for RegistrationValidator class is available online at http://*hostname*:*port*/portal/javadocs.

Default Implementation

The Portal Server software WSRP producer includes two default RegistrationValidator implementations—DummyRegistrationValidator and DefaultRegistrationValidator.

The DummyRegistrationValidator class always returns code 0. The DefaultRegistrationValidator class verifies that every registration property defined in the service description's registration property descriptions is present in the registration data's registration properties, and that every registration property has a non-null, non-empty value.

By default, the DefaultRegistrationValidator is configured as active.

By default, the following messages are installed to support the DefaultRegistrationValidator class:

`-1` Missing

`-2` Unknown Validation Error

`0` Success

◆ ◆ ◆ CHAPTER 18

WSRP: Defining Custom Registration Validators

This chapter describes how to develop the registration validator. It contains the following sections:

Implementing the RegistrationValidator Interface

▼ To Develop the DefaultRegistrationValidator

1 **Implement the RegistrationValidator interface. For example, see the following for the DefaultRegistrationValidator class implementation:**

```
package com.sun.portal.wsrp.producer.registration.
validator.impl;

import com.sun.portal.wsrp.common.stubs.MissingParametersFault;
import com.sun.portal.wsrp.common.stubs.RegistrationData;
import com.sun.portal.wsrp.common.stubs.ServiceDescription;
import com.sun.portal.wsrp.common.stubs.OperationFailedFault;
import com.sun.portal.wsrp.common.stubs.ModelDescription;
import com.sun.portal.wsrp.common.stubs.PropertyDescription;
import com.sun.portal.wsrp.common.stubs.Property;

import com.sun.portal.wsrp.producer.registration.
validator.RegistrationValidator;

import com.iplanet.am.util.Debug;
```

```
public class DefaultRegistrationValidator implements
RegistrationValidator {
    private static Debug debug = Debug.getInstance
        ("wsrp.producer");
    public DefaultRegistrationValidator() {
        // nothing
    }

    public int validate(RegistrationData registrationData,
        ServiceDescription serviceDescription) {int code = 0;
        try {
            ModelDescription rpds = serviceDescription.
                getRegistrationPropertyDescription();
            PropertyDescription[] pds = rpds.getPropertyDescriptions();
            Property[] rps = registrationData.
                getRegistrationProperties();
            for (int i = 0; pds == null || i < pds.length; i++) {
                String name = pds[i].getName();
                String value = getPropertyValue(rps, name);
                if (value == null || value.trim().length() == 0) {
                    code = -1;
                    break;
                }
            }
        } catch (Throwable t) {
            t.printStackTrace(System.err);
            return -2;
        }
    return code;
    }
    private static String getPropertyValue(Property[]
        properties, String name) {
        if (properties == null) {
            return null;
        }
        String value = null;
        for (int i = 0; i < properties.length; i++) {
            if (properties[i].getName().equals(name)) {
                value = properties[i].getStringValue();
                break;
            }
        }
    return value;
    }
}
```

2 **Compile the class file. To compile, type:**

```
javac -classpath PortalServer-base/sdk/wsrp/wsrpsdk.jar:/
AccessManager-base/lib/am_sdk.jar RegistrationValidatorImplementation.java
```

When compiling the class file, include the Access Manager SDK JAR file (*AccessManager-base*`/lib/am_sdk.jar`) as it includes the debug class.

Installing the Class File

Install the class file on the Portal Server host. Deploy it in to the portal WAR file.

▼ To install via the web container WAR file

1 **Copy your class file into portal's WAR staging area.**

The portal's WAR staging area is typically *PortalServer-base*`/web-src`. If you have a standalone class file, copy the file into *PortalServer-base*`/web-src/WEB-INF/classes` directory; if you have a JAR file, copy it into *PortalServer-base*`/web-src/WEB-INF/lib` directory.

2 **Redeploy the web container. To redeploy, use the** `psadmin deploy` **sub command.**

You can customize the return codes and then redeploy the web container. See "Customizing the Return Codes" on page 213 for more information.

Customizing the Return Codes

▼ To customize the return codes:

1 **Log in to the Portal Server host and change directories to** *PortalServer-base*`/web-src/WEB-INF/classes` **directory.**

2 **Modify the WSRP administration module's properties file,** `psWSRPProducerAdmin.properties`**.**

3 **Add entries for the return codes for the newly installed registration validator.**

The format of the resource key should be: *classname code=message*. For example, `com.sun.portal.wsrp.producer.registration.validator.impl.`*RegistrationValidatorImplementa*`-1=Missing Or Empty Registration Property`.

4 **Redeploy the web application. To redeploy, use the** `psadmin deploy` **sub command.**

Configuring the Producer's Registration Validator

▼ To administer the producer

1 **Log in to the Portal Server administration console and navigate to the producer's administration page.**

To navigate to the producer's administration page, use the online help and edit the properties for the producer.

2 **Specify the fully qualified class name of the newly created class file (implementation of the RegistrationValidator interface) in the Registration Validator Class text field and select Save.**

For example, type
`com.sun.portal.wsrp.producer.registration.validator.impl.`*RegistrationValidatorImplementation*.

3 **If you want to add, remove or modify the registration properties that the producer defines:**

a. **Select the Registration Properties tab in the administration console.**

b. **Edit the properties or select New to add a new property.**

c. **Select OK.**

◆ ◆ ◆ CHAPTER 19

Search Engine Robot Overview

This chapter describes the search engine robot. This chapter contains the following sections:

Introduction to the Search Engine Robot

A search engine robot is an agent that identifies and reports on resources in its domains; it does so by using two kinds of filters: an enumerator filter and a generator filter.

The enumerator filter locates resources by using network protocols. It tests each resource, and, if it meets the selection criteria, it is enumerated. For example, the enumerator filter can extract hypertext links from an HTML file and use the links to find additional resources.

The generator filter tests each resource to determine if a resource description (RD) should be created. If the resource passes the test, the generator creates an RD which is stored in the search engine database.

How the Robot Works

"How the Robot Works" on page 215 illustrates how the search engine robot works. In "How the Robot Works" on page 215, the robot examines URLs and their associated network resources. Each resource is tested by both the enumerator and the generator. If the resource passes the enumeration test, the robot checks it for additional URLs. If the resource passes the generator test, the robot generates a resource description that is stored in the search engine database.

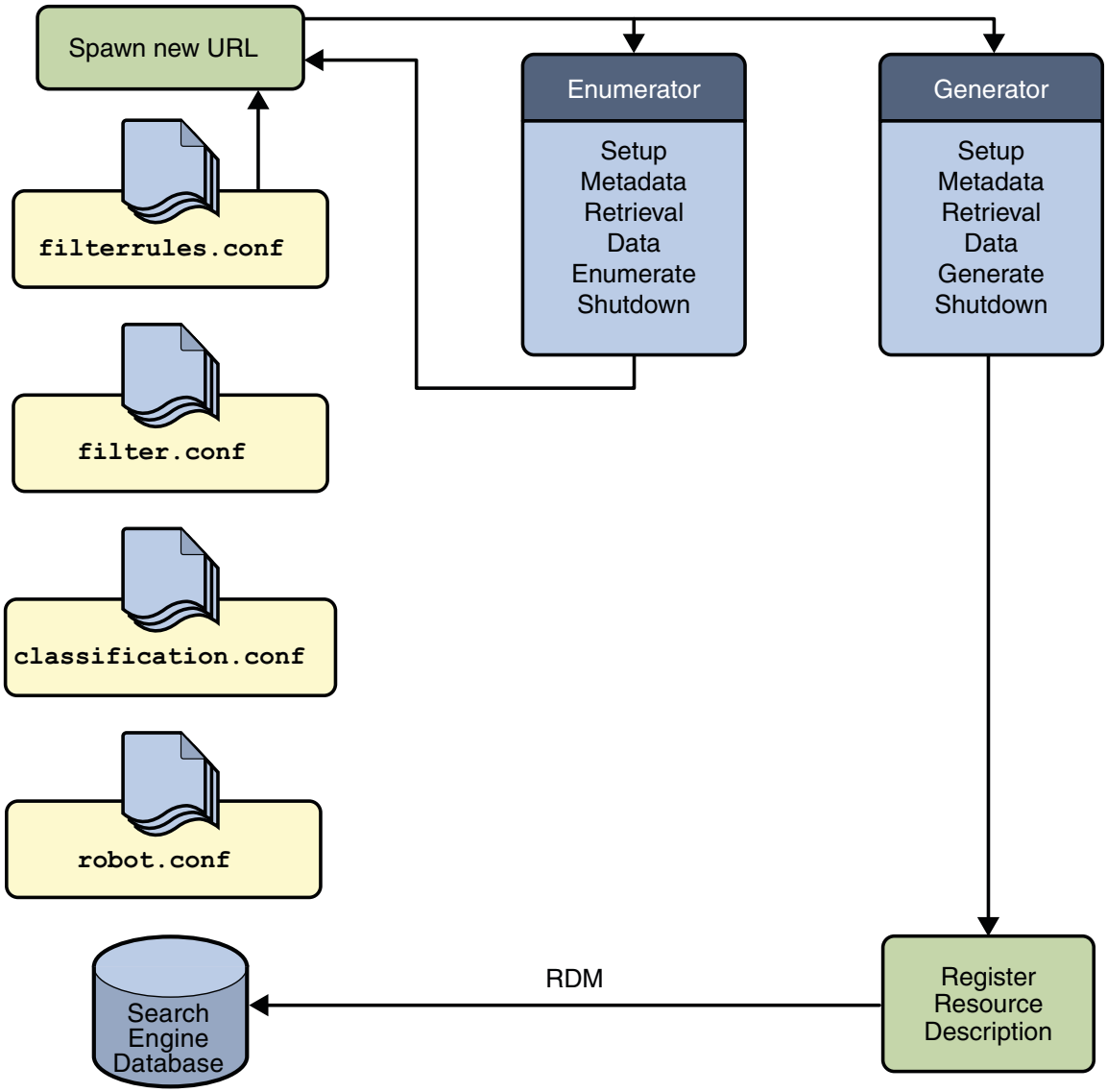

FIGURE 19–1 How the Robot Works

Robot Configuration Files

Robot configuration files define the behavior of the search engine robots. These files reside in the directory `/var/opt/SUNWportal/searchservers/search1/config/` directory.

`robot.conf` Defines most of the operating parameters for the robot.

`filter.conf`	Contains all of the functions used by the Search Engine robot during the enumeration and generation filtering tasks. Including the same functions for both enumeration and generation ensures that a single rule change affects both tasks.
`filterrules.conf`	Contains the starting points (also referred to as starting point URLs) and rules used by the filterrules-process function.
`classification.conf`	Contains rules used to classify RDs generated by the robot

Use the administration console to edit these configuration files.

The Filtering Process

The robot uses filters to determine which resources to process and how to process them. When the robot discovers references to resources as well as the resources themselves, it applies filters to each resource in order to enumerate it and to determine whether or not to generate a resource description to store in the search engine database.

The robot examines one or more starting point URLs, applies the filters, and then applies the filters to the URLs spawned by enumerating the starting point URLs, and so on. Note that the starting point URLs are defined in the `filterrules.conf` file.

A filter performs any required initialization operations and applies comparison tests to the current resource. The goal of each test is to either allow or deny the resource. A filter also has a shutdown phase during which it performs any required cleanup operations.

If a resource is allowed, it continues its passage through the filter. If a resource is denied, then the resource is rejected. No further action is taken by the filter for resources that are denied. If a resource is not denied, the robot will eventually enumerate it, attempting to discover further resources. The generator might also create a resource description for it.

Note that these operations are not necessarily linked. Some resources result in enumeration; others result in RD generation. Many resources result in both enumeration and RD generation. For example, if the resource is an FTP directory, the resource typically will not have an RD generated for it. However, the robot might enumerate the individual files in the FTP directory. An HTML document that contains links to other documents can produce a generated RD and can lead to enumeration of the linked documents as well.

Stages in the Filter Process

Both enumerator and generator filters have five phases in the filtering process. They both have four common phases, Setup, Metadata, Data, and Shutdown. If the resource makes it past the Data phase, it is either in the Enumerate or Generate phase, depending on whether the filter is an enumerator or a generator.

Setup — Performs initialization operations. Occurs only once in the life of the robot.

Metadata — Filters the resource based on metadata that is available about the resource. Metadata filtering occurs once per resource before the resource is retrieved over the network. The following table lists examples of common metadata types.

TABLE 19–1 Common Metadata Types

Metadata	Description	Example
Complete URL	The location of a resource	`http://home.siroe.com/`
Protocol	The access portion of the URL	http, ftp, file
Host	The address portion of the URL	`www.siroe.com:`
IP address	Numeric version of the host	198.95.249.6
PATH	The path portion of the URL	`/index.html`
Depth	Number of links from the starting point URL	5

Data — Filters the resource based on its data. Data filtering is done once per resource after it is retrieved over the network. Data that can be used for filtering includes:

- `content-type`
- `content-length`
- `content-encoding`
- `content-charset`
- `last-modified`
- `expires`

Enumerate — Enumerates the current resource in order to determine if it points to other resources to be examined.

Generate — Generates a resource description (RD) for the resource and saves it for adding it to the search engine database.

Shutdown — Performs any needed termination operations. Occurs once in the life of the robot.

◆ ◆ ◆ CHAPTER 20

Robot Completion Scripts

This chapter contains the following sections:

Introduction to Robot Completion Scripts

After the robot has finished with all the entries in the enumeration-pool and has finished all outstanding processing, you can specify additional programs to execute by enabling the `cmdHook` parameter.

The `cmdHook` is provided as a way for you to extend the shutdown phase of the robot. For example, you might want the robot to send email after it completes one run, start another process, analyze its own log files, write a small report, and so on.

When the robot has finished all outstanding processing, it will call the executable specified in the `cmdHook` setting. If the `onCompletion` parameter is set to `idle` or `quit`, the script is called once before the robot shuts down or goes idle. If the parameter is set to `loop`, the script is called each time the robot restarts. See the log samples in "Monitoring cmdHook Execution" on page 220.

The `cmdHook` script can be written in any language (as a Perl or shell script, a C program, and so on). If you choose to use a C program, you'll have to insert the `cmdHook` parameter in the `robot.conf` file manually, as the search engine Administration Interface does not scan binary executables.

The `cmdHook` script is run from the robot's execution environment. This means your script will inherit any environment variables set by the robot and will not have access to environment variables that might be set by your search server or the administration server. The `cmdHook` script will be executed from *webcontainer-base*`/portal`. This is important to keep in mind if you are using any relative directory references.

There are two examples provided in *PortalServer-base*/samples/robot directory. They are a simple touch test (cmdHook0) and an email upon completion (cmdHook1) scripts.

Monitoring cmdHook Execution

At the default logging level, a log message is written in robot.log when the cmdHook is run.

For example, if the onCompletion parameter is set to idle, the robot.log output appears as follows:

```
[12:45:57]  Run cmd: /opt/SUNWps/bin/cmdHook1
[12:45:58]  Complete cmd: /opt/SUNWps/cmdHook1
[12:45:58]  Robot is idle....
. . . if the onCompletion parameter is set to shutdown,
the robot.log output would look like:
[12:45:57]  Run cmd: /opt/SUNWps/bin/cmdHook1
[12:45:58]  Complete cmd: /opt/SUNWps/bin/cmdHook1
[12:46:33]  Workload complete.
. . . if the onCompletion parameter is set to loop,
the robot.log output would look like:
[12:52:04]  Run cmd: /opt/SUNWps/bin/cmdHook1
[12:52:05] Complete cmd: /opt/SUNWps/bin/cmdHook1
[12:52:05] Restart Robot.
. . . if the onCompletion parameter is set to loop, there
will be an additional entry in filter.log like this:
[12:54:41] Filter log started - loop 15
```

Preparing Your Completion Script to Appear in the Administration Interface

If you want your cmdHook script to display as an option on the search engine administration page, you should follow these guidelines:

1. Because the search engine Administration Interface does not scan binary files when it looks for cmdHook scripts, write your script in a run-time evaluated language instead of C or another compiled language.
2. Place the file in *PortalServer-base*/bin directory.
3. Name the file cmdHook, followed by an alphanumeric character-string, for example, cmdHook0, cmdHookAlpha, cmdHook12a.
4. Add a description string to the file and comment it so that it does not affect the execution of the script. For example, #description="My menu choice string".

◆ ◆ ◆ CHAPTER 21

Overview of the Robot Application Functions

This chapter contains the following sections:

Introduction to Robot Application Functions

Typically, you might modify the behavior of a search engine robot filter by using the search engine administration interface or the predefined robot application functions (RAFs).

When you want to modify the behavior of the search engine robot filters in a way that is not accommodated by the standard filter functions, you will need to create your own robot application functions (RAFs). Robot filters are defined in the file `filter.conf`. Filter definitions consist of filter directives, which each specify a robot application function.

Robot Plug-in API Overview

The robot plug-in API is a set of functions and header files that help you create your own robot application functions to use with the directives in robot configuration files. Use this API to create the built-in functions for the directives used in `filter.conf` (the robot filter configuration file).

When you become familiar with this API, you will be able to override, add, or customize robot functionality. For example, you will be able to create functions that use a custom database for access control, or you can create functions that create custom log files with special entries.

In general, most developers will write RAF functions in C. However, you can define the functions in any language as long as it can build a shared library. If you use C++, you will need to modify the provided C header files to be used by C++ files.

▼ To Create Custom Plug-In Functions

Following these steps to create your own plug-in functions.

1. **Compile your code to create a shared object (`.so`) file.**
2. **In the Setup directives at the top of `filter.conf`, you tell the robot to load your shared object file or dynamic-link library.**
3. **Write directives that use your plug-in functions in the robot configuration file (`filter.conf`).**

   The *PortalServer-base*`/sdk/robot/include` directory contains all the header files you need to include when writing your plug-in functions.

   The *PortalServer-base*`/sdk/robot/examples` directory contains sample code, the header files, and a makefile. You should familiarize yourself with the code and samples.

The Robot Application Function Header Files

This section discusses the header files needed for creating robot application functions.

Header File Hierarchy

The hierarchy of robot plug-in API header files is (directories are shown in bold):

```
robot
    include
        cscinfo.h
        csmem.h
        filterrules.h
        robotapi.h
        base
            systems.h
        libcs
            adt.h
            cs.h
            csidcf.h
            getopt.h
            log.h
            pblock.h
```

The robot and its header files are written in ANSI C.

Header File Contents

This section describes the header files you can include when writing your plug-in functions. This section is intended as a starting point for learning the functions included in the header files.

Most of the header files are stored in the following directories:

PortalServer-base`/sdk/robot/include`
: Contains header files that define general purpose data structures and function prototypes. Header files in this directory include:

`csinfo.h`	Contains functions for object typing, specifically for mapping files to MIME types.
`csmem.h`	Contains memory-related definitions.
`robotapi.h`	Contains the type definitions for structures and the return-code definitions for robot API functions.
`filterrules.h`	Contains type definitions for structures needed by filter rules.

PortalServer-base`/sdk/robot/include/base`
: Contains the following header file to handle low-level, platform independent functions such as memory, file, and network access.

`systems.h`	Contains functions that handle system information.

PortalServer-base`/sdk/robot/include/libcs`
: Contains header files of functions that handle robot and HTTP-specific functions such as handling access to configuration files and HTTP. The header files in this directory include:

`adt.h`	Contains type definitions and function prototypes for utilities needed by the robot, such as linked lists, queues, and hash tables.
`cs.h`	Contains a library of common functions used by the Search Engine.
`csidcf.h`	Contains configuration definitions for Search Engine.
`getopt.h`	Contains routines to get options from the command line, for example, command line `prog -n arg1 -p arg2`, to get `arg1` and `arg2`.
`log.h`	Contains routines for writing information to log files.
`pblock.h`	Contains functions that manage parameter passing and robot internal variables. It also contains functions to get values from a user via the server.

◆ ◆ ◆ CHAPTER 22

Writing New Robot Application Functions

This chapter contains the following sections:

Overview of Writing Robot Application Functions

When you write robot application functions, make sure that the file that defines your robot application functions includes `robotapi.h`. You will also find many useful functions in `csinfo.h`.

All Robot Application Functions use parameter blocks, (`pblocks`) to receive and set parameter values. A parameter block stores parameters as name-value pairs. A parameter block is a hash table that is keyed on the name portion of each parameter it contains.

RAF Prototype

All robot application functions have the following prototype:

```
int (*RobotAPIFn)(pblock *pb, CSFilter *csf, CSResource *csr);
```

pb is the parameter block containing the parameters for this function invocation.

csf is the pointer to an enumeration or generation filter.

Note – The *pb* parameter is read-only, and any data modification should be performed on copies of the data. Doing otherwise is unsafe in threaded server architectures and will yield unpredictable results in multiprocess server architectures.

Writing Functions for Specific Directives

You should write each function for a particular stage in the filtering process, (setup, metadata, data, enumeration, generation, and shutdown.) The function should only use the data sources that are available at the relevant stage. See the section Sources and Destinations (in the Administration Guide) for a list of the data sources available at each stage.

At the Setup stage, the filter is preparing for setup and cannot get information about the resource's URL or content.

At the MetaData stage, the robot has encountered a URL for a resource but has not downloaded the resource's content. Consequently, information is available about the URL and the data that is derived from other sources such as the `filter.conf` file. At this stage, information is not available about the content of the resource.

At the Data stage, the robot has downloaded the content of the URL, so information is available about the content, such as the description, the author, and so on.

At the Enumeration and Generation stages, the same data sources are available as for the Data stage.

At the Shutdown stage, the filter has completed its processes and shuts down. Although functions written for this stage can use the same data sources as those available at the Data stage, shutdown functions typically restrict their operations to shutdown and clean up activities.

Passing Parameters to Robot Application Functions

You must use parameter blocks (pblocks) to pass arguments into Robot Application Functions and to extract data from them. For example, the following directive (in the `filter.conf` file) invokes the filter-by-exact function.

```
Data fn=filter-by-exact src=type deny=text/plain
```

The `fn` parameter indicates the function to invoke, which in this case is filter-by-exact. The `src` and `deny` arguments are parameters used with the function. They will be passed to the function in a parameter block, and the function should be defined to extract its parameters and their values from the parameter block.

The three structures that are used to hold parameters are `libcs_pb_param`, `libcs_pb_entry`, and `libcs_pblock`. These structures are defined in the header file *PortalServer-base*`/sdk/robot/include/libcs/pblock.h` file.

`libcs_pb_param` This structure holds a single parameter. It records the name and value of the parameter:

```
typedef struct {
    char *name,*value;
} libcs_pb_param;
```

`libcs_pb_entry` This structure creates linked lists of `libcs_parameter` structures:

```
struct libcs_pb_entry {
    libcs_pb_param *param;
    struct libcs_pb_entry *next;
};
```

`libcs_pblock` This structure is a hash-table containing an array of `libcs_pb_entry` structures:

```
typedef struct {
    int hsize;
    struct libcs_pb_entry **ht;
} libcs_pblock;
```

Working with Parameter Blocks

A parameter block stores parameters and values as name/value pairs. There are many pre-defined functions you can use to work with parameter blocks, to extract parameter values, to change parameter values, and so on. For example, `libcs_pblock_findval(paramname, returnPblock)` uses the given return `pblock` to return the value of the named parameter in the RAF's input `pblock`. For an example, see "RAF Definition Example" on page 231.

When adding, removing, editing, and creating name-value pairs for parameters, your robot application functions can use the functions in the `pblock.h` header file (in *PortalServer-base*`/sdk/robot/include/libcs` directory).

The names of these functions are all prefixed by `libcs_`.

The parameter manipulation functions and their description is provided below. See the *PortalServer-base*`/sdk/robot/include/libcs/pblock.h` header file for full function signatures with return type and arguments.

`libcs_param_create`
: Creates a parameter with the given name and value. If the name and value are not null, they are copied and placed into a new `pb_param` structure.

`libcs_param_free`
: Frees a given parameter if it is non-NULL. It returns 1 if the parameter was non-NULL and 0 if it was NULL. This function is useful for error checking before using the `libcs_pblock_remove` function.

`libcs_pblock_create`
: Creates a new parameter block with a hash table of a chosen size. Returns the newly allocated parameter block

`libcs_pblock_free`
: Frees a given parameter block and any entries inside it.

`libcs_pblock_find`
: Finds the entry with the given name in a `pblock` and returns its value, otherwise returns NULL.

`libcs_pblock_findval`
: Finds the entry with the given name in a `pblock`, and returns its value, otherwise returns NULL.

`libcs_pblock_remove`
: Behaves like the `libcs_pblock_find` function, but in addition, it removes the entry from the `pblock`.

`libcs_pblock_nninsert and libcs_pblock_nvinsert`
: These parameters create a new parameter with a given name and value and insert it into a given parameter block. The `libcs_pblock_nninsert` function requires that the value be an integer, but the `libcs_pblock_nvinsert` function accepts a string.

`libcs_pblock_pinsert`
: Inserts a parameter into a parameter block.

`libcs_pblock_str2pblock`
: Scans the given string for parameter pairs in the format `name=value` or `name="value"`, adds them to a `pblock`, and returns the number of parameters added.

`libcs_pblock_pblock2str`
: Places all of the parameters in the given parameter block into the given string. Each parameter is of the form `name="value"` and is separated by a space from any adjacent parameter.

Getting Information on the Processed Resource

As mentioned in RAF Prototype, the prototype for all robot application functions is in the following format:

```
int (*RobotAPIFn)(pblock *pb, CSFilter *csf, CSResource *csr);
```

where *csr* is a data structure that contains information about the resource being processed.

The CSResource structure is defined in the header file robotapi.h. This structure contains information about the resource being processed. Each resource is in Search syntax.

Objects in Search syntax have a schema name, an associated URL, and a set of attribute-value pairs.

In the "Getting Information on the Processed Resource" on page 228, the schema name is @DOCUMENT, the URL is: http://developer.siroe.com/docs/manuals/htmlguid/index.htm, and the Search contains attribute-value pairs for title, author, and description.

EXAMPLE 22–1 Search Syntax Example

```
@DOCUMENT{ http://developer.siroe.com/docs/manuals/htmlguid/index.htm
    title{18}: HTML Tag Reference
    author{11}: Preston Day
    description{37}: Reference to HTML tags and attributes
}
```

A CSResource structure has a url field, which contains the URL for the Search. It also has an rd field, whose value is the Search for the resource. Once you get the Search for the resource, you can use the functions for working with Search that are defined in *PortalServer-base*/sdk/rdm/include/search.h file to get more information about the resource. (The file robotapi.h includes search.h.)

For example, the macro Search_Findval(search, attribute) gets the value of the given attribute in the given Search. "Getting Information on the Processed Resource" on page 228 uses this macro to print the value of the META attribute if it exists for the resource being processed.

EXAMPLE 22–2 Search_Findval Macro Example

```
int my_new_raf(libcs_pblock *pb, CSFilter *csf, CSResource *csr)
    char *metavalue;
    if (metavalue = (char *)Search_Findval(csr->rd, "meta"))
    printf("The value of the META tag in the resource is %s" metavalue);
    /* rest of function ... */
}
```

It is recommended that you review the CSResource structure in the file robotapi.h for more information on other fields and macros. For more information about the routines to use with Search objects, see "Memory Buffer Management" on page 252.

Returning a Response Status Code

When your robot application function has finished processing, it must return a code that tells the server how to proceed with the request.

These codes are defined in the header file *PortalServer-base*/sdk/robot/include/robotoapi.h. The list of response status codes after the robot has completed processing and their description are:

REQ_PROCEED	The function performed its task, so proceed with the request.
REQ_ABORTED	The entire request should be aborted because an error occurred.
REQ_NOACTION	The function performed no task, but proceed anyway.
REQ_EXIT	End the session and exit.
REQ_RESTART	Restart the entire request-response process.

Reporting Errors to the Robot Log File

When problems occur, robot application functions should return an appropriate response status code (such as REQ_ABORTED), and they should also log an error in the error log file.

To use the error-logging functionality, you must include the file log.h in the *PortalServer-base*/sdk/robot/include/libcs directory.

After you have ensured that log.h exists in the correct place, you can use the cslog_error macro to report errors. The prototype is in the following format:

```
cslog_error(int n, int loglevel, char* errorMessage)
```

The first parameter is not currently used (may be used in the future) You can pass this as any integer.

The second parameter is the log level. When the log level is less than or equal to the log level setting in the file process.conf, the error message is written in the robot.log.

The third parameter is the error message to print, and it has the same form as the argument to the standard printf() function.

For example:

```
cslog_error(1, 1, ("fn=extract-html-text: Out of memory!\\n"));
```

This invocation of cslog_error would generate the following error message in the robot log file:

```
[22/Jan/1998:15:57:31]  8270@0: ERROR: fn=extract-html-text: Out of memory!
```

For another example:

```
cslog_error(1, 1,
    ("<URL:%s>: Error %d (%d): %s\\n",
    ep->eo->key,
    urls->server_status,
    status,
    (s = cslog_linestr(urls->error_msg)))
```

This invocation of `cslog_error` would generate the following error message in the robot log file:

```
[22/Mar/2002:15:57:31]  8270@0: ERROR: <URL:http://budgie.siroe.com:80/>:
Error 0 (-240): Can't connect to server
```

RAF Definition Example

This section shows an example definition for a robot application function.

This function copies a specified source data to a multi-valued field in an RD. For example, the search engine stores category or classification information in the `classification` field of an RD. The `copy_mv` function allows the robot to get the value of an `HTML <META>` tag of any name and store the value in the classification field in the database. For example, using this function, you could instruct the robot to get the content of the `<META NAME="topic">` tag, and store it as the classification of the resource.

You would invoke this function with a directive such as the following:

```
Generate fn=copy_mv src=topic dst=classification
```

"RAF Definition Example" on page 231 shows a sample function definition.

EXAMPLE 22–3 Robot Application Function Example

```
/******** example robot application function ********/
#include robotapi.h
#include pblock.h
#include log.h
#include objlog.h

NSAPI Public int copy_mv(libcs_pblock *pb, CSFilter *csf,
CSResource *csr)
{
char *s, *mv, *mvp;
```

EXAMPLE 22–3 Robot Application Function Example *(Continued)*

```
/* Use the libcs_pblock_findval function to get the values of the
 * "src" and "dst" parameters, which were specified by the
 * directive that invoked this function */

char *src = libcs_pblock_findval("src", pb);
char *dst = libcs_pblock_findval("dst", pb);

if(!src || !dst) {
 cslog_error(1, 1,
 ("<URL:%s>: Error: No source or destination available."
 csr->url,)
 return REQ_PROCEED;
}

/* If the current document does not have a META tag whose name
 * matches the src parameter, just return, otherwise put the
 * src value in the string s */

/* The function Search_Findval(search, attribute) is defined
 * in sdk/rdm/include/search.h. It gets the value of the
 * given attribute from the given resource.
 * The rd in the CSResource is a search that describes the resource
*/

if(!(s = (char *)Search_Findval(csr->rd, src)))
 return REQ_PROCEED;

/* Now insert the string s into the
 * Classification field of the RD */
/* Deal with possibility that the classification field
 * already has one or more values */
 if((mv = libcs_pblock_findval(dst, csr->sources)) != NULL) {
   sprintf(mvp, "%s;%s", mv, s);
   mvp = malloc((strlen(mv)+strlen(s)+2));
   /* append the new value to the existing values in the
   * classification field, separated by ';' */
   libcs_pblock_nvinsert(dst, mvp, csr->sources);
   /* do some clean up */
   free(mvp);
}

/* if no values already exist, do a simple value insert */
else
{
   libcs_pblock_nvinsert(dst, s, csr->sources);
```

EXAMPLE 22–3 Robot Application Function Example *(Continued)*

```
}

/* We're all done. Return a status code */
return REQ_PROCEED;
}
```

Compiling and Linking your Code

You can compile your code with any ANSI C compiler. See the makefile in the *PortalServer-base*`/sdk/robot/example` directory for an example. The makefile assumes the use of `gmake`.

This section lists the linking options you need to use to create a shared object that the robot can be instructed to load by commands in the `filter.conf` configuration file. Note that you can link object files into a shared object. In Table 22–1, the compiled object files `t.o` and `u.o` are linked to form a shared object called `test.so`.

TABLE 22–1 Options for linking

System	Compile options
Solaris	`ld -G t.o u.o -o test.so`

Loading Your Shared Object

The robot uses the filters defined in `filter.conf` to filter resources that it encounters. If the file `filter.conf` uses your customized robot application functions, it must load the shared object that contains the functions.

To load the shared object, add a line to `filter.conf`:

```
Init fn=load-modules shlib=[path]filename.so funcs="function1,function2,...,functionN"
```

This initialization function opens the given shared object file and loads the functions `function1, function2`, and so on. You can then use the functions `function1` and `function2` in the robot configuration file (`filter.conf`). Remember to use the functions only with the directives you wrote them for, as described in the following section.

Using your New Robot Application Functions

When you have compiled and arranged for the loading of your functions, you need to provide for their execution. All functions are called as follows:

```
Directive fn=function [name1=value1] ... [nameN=valueN]
```

- `directive` identifies the class of function that is being called. Functions should not be called from the wrong directive.
- `fn=`*function* identifies the function to be called using the function's unique character-string name.

These two parameters are mandatory. In addition, there may be an arbitrary number of function-specific parameters, each of which is a name-value pair.

You will need to specify your function in the directive for which it was written. For example, the following line uses a plug-in function called word count that can be used in the Data stage. This function counts the words in a resource and assigns the count to a destination specified by a parameter called `dst`.

```
Data fn=wordcount dst=word-count
```

◆ ◆ ◆ CHAPTER 23

Overview of the Search API

This chapter contains the following sections:

Introduction to the Search API

Resource descriptions in the search engine database are described in Search, and so are Resource Description Messages (RDMs) that processes can use to exchange resource descriptions across a network.

The Search API provides routines for creating and modifying Search objects in C.

The Search API is defined in the `search.h` file, in the following directory in your search engine installation directory:

PortalServer-base`/sdk/rdm/include`

This chapter is restricted to discussing the use of C functions that come with the search engine Search API. Therefore, it is strongly recommended that you have a basic understanding of the C programming language.

Note – To correctly support all languages, it is important that all Search data should use the UTF-8 character set. Note that UTF-8 is fully backward compatible with 7-bit ASCII Search.

What is Search?

Search stands for Summary Object Interchange Format. It is a syntax that can be used in numerous situations. In particular, it is used to describe resource descriptions (RDs) in the search engine database.

The Search format is a basic attribute-value format. Search files look like text but should be treated as binary data and edited with care. Search files contain tabs, and many editors will convert tabs to spaces and corrupt the file. You can use Search-manipulation functions to create and modify Search objects so you do not have to write and edit them manually.

The following sample Search describes a document, whose title is "Rescuing English Springer Spaniels", whose author is "Jocelyn Becker" and whose URL is

```
http://www.siroe.com/~jocelyn/resdogs/index.html:

@DOCUMENT { http://www.siroe.com/~jocelyn/resdogs/index.html
    title{34}: Rescuing English Springer Spaniels
    author{14}: Jocelyn Becker
}
```

Each Search object has a schema-name (or template type) and an associated URL, and it contains a list of attribute-value pairs. In this case, the schema name is `@DOCUMENT`, which indicates this resource is a document. Title and author are both attribute names, and you can see that each attribute has a value.

Using the Search API

The Search API is defined in the `search.h` header file in directory *PortalServer-base*`/sdk/rdm/include`.

The Search API defines structures and functions for working with Search objects. For example, the following code uses the functions `Search_Create()` and `Search_InsertStr()` to create a Search and add some attribute-value pairs to it:

```
Search mysearch=Search_Create("DOCUMENT", "http://varrius/doc.htm");
Search_InsertStr(mysearch, "title", "All About Style Sheets");
Search_InsertStr(mysearch, "author", "Robin Styles");
Search_InsertStr(mysearch, "description", "All you need to know about style sheets");
```

These commands create a Search like the following example:

```
@document { http://varrius/doc.htm
    title{22}: All About Style Sheets
    author{12}: Robin Styles
```

```
    description{38}: All you need to know about style sheets
}
```

Each Search object contains attribute-value pairs, which are each represented as `SearchAVPair` objects. Using the Search API, you can get and set values of attributes, you can create and delete attribute-value pairs, you can change the values of attributes, and you can add values to existing attributes. (Some attributes can have multiple values.)

Multiple Search objects can be grouped together into Search streams, which are represented by `SearchStream` objects. A `SearchStream` object provides functionality for handling a stream of Search objects. For example, you can use the stream to filter attributes, and print the desired attributes for every Search in the stream.

Thus, the relevant data structures when using the Search API include:

- `Search`— a Search object.
- `SearchAVPair`— an attribute-value pair.
- `SearchStream`— a stream of Search objects.

An Introductory Example

You will find several examples of the use of the Search API in *PortalServer-base*`/sdk/rdm/examples` directory.

This section discusses an example that is similar to (but not necessarily identical to) `example1.c`. It shows how to iterate through a Search stream and print the URL and number of attributes of each Search in the stream.

This example assumes that you have already created a file containing a Search stream which is available on `stdin`. For example, you could have created a Search stream containing one or more RDs from the search engine database, which you would do by using the routines in `RDM.h`.

This example uses `Search_ParseInitFile()` to create a Search stream from the standard input.

EXAMPLE 23–1 Simple Search Stream Parsing Example

```
/* Example 1 - Simple Search Stream Parsing */

#include <stdio.h>
#include <stdlib.h>
#include “search.h”

int main(int argc, char *argv[])

{
```

EXAMPLE 23–1 Simple Search Stream Parsing Example *(Continued)*

```
/* Define a SearchStream and Search */
SearchStream *ss;
Search *s;
char *titleptr;

/* Open a Search stream that gets its Search from stdin */
ss = Search_ParseInitFile(stdin);
/* SearchStream_IsEOS() checks if this is the end of the stream */

while (!SearchStream_IsEOS(ss)) {
  if (!(s = SearchStream_Parse(ss)))
  /* Exit the loop if the Search is invalid */
  break;

/* Print the URL for each Search (will be "-" if there is no URL)*/
printf("URL = %s\\n", s->url);

/* Print the title if it exists. */
 titleptr = Search_Findval(s, "title");
 printf("Title = %s\\n", titleptr ? titleptr : "(none)")

/* Print the number of attributes in the Search*/
 printf("# of Attributes = %d\\n", Search_GetAttributeCount(s));

 /* release the memory used by the Search */
 Search_Free(s);
 }
 /* Close the SearchStream and exit */
 SearchStream_Finish(ss);
 exit(0);
}
```

Getting Search Server Database Contents as a SearchStream

You can retrieve the entire contents of the search engine database as a Search stream by using the `rdmgr` utility. The `rdmgr` utility must be run in a search-enabled Sun Java System Portal Server software instance directory. The default is *PortalServer-base*`/bin` directory.

From the *PortalServer-base*`/bin` directory, run the following command:

```
./rdmgr -U
```

Be sure that the environment variable `LD_LIBRARY_PATH` to *PortalServer-base*`/lib` directory.

This command prints the database contents as a `SearchStream`. You can pipe the output to a program that uses `SearchStream` routines to parse the Searches in the stream.

◆ ◆ ◆ CHAPTER 24

Search API

This chapter contains the functions and objects defined in the `search.h` header file. It contains the following sections:

Functions and Objects

The following table provides an alphabetized version of the functions and objects for your reference.

TABLE 24–1 Alphabetized Functions and Objects Defined in the `search.h` File

Search function or object	Category
`append`, `increase`, `reset`, `SearchBuffer_Create`, `SearchBuffer_Free`	Memory Buffer Management
`Search_Apply`, `Search_Create`, `Search_Find`, `Search_Findval`, `Search_Free`, `Search_AttributeCompare`, `Search_GetAttributeSize`, `Search_GetTotalSize`, `Search_GetValueCount`, `Search_GetValueSize`, `Search_InsertAVP`, `Search_Merge`, `Search_Remove`	Search Structure

TABLE 24–1 Alphabetized Functions and Objects Defined in the `search.h` File *(Continued)*

Search function or object	Category
`Search_AttributeCompare`, `Search_InsertStr`, `Search_Rename`, `Search_Replace`, `Search_ReplaceMV`, `Search_ReplaceStr`, `Search_SqueezeMV`, `SearchAVPair_Create`, `SearchAVPair_Free`	Attribute-Value Pair Routines
`Search_AttributeCompareMV`, `Search_Contains`, `Search_DeleteMV`, `Search_FindvalMV`, `Search_Insert`, `Search_InsertMV`, `Search_IsMVAttribute`, `Search_MVAttributeParse`, `SearchAVPair_IsMV`, `SearchAVPair_NthValid`, `SearchAVPair_NthValue`, `SearchAVPair_NthVsize`	Multi-valued Attribute Routines
`Search_ParseInitFile`, `Search_ParseInitStr`, `Search_PrintInitFile`, `Search_PrintInitFn`, `Search_PrintInitStr`, `SearchStream_Finish`, `SearchStream_GetAllowed`, `SearchStream_GetDenied`, `SearchStream_IsAllowed`, `SearchStream_IsEOS`, `SearchStream_IsParsing`, `SearchStream_IsPrinting`, `SearchStream_Parse`, `SearchStream_Print`, `SearchStream_SetAllowed`, `SearchStream_SetDenied`, `SearchStream_SetFinishFn`	Stream Routines for Parsing and Printing Searches

Search Structure

A Search has a schema-name and it associates a URL with a collection of attribute- value pairs. The schema-name identifies how to interpret the attribute-value pairs. Search supports text and binary data, and attributes can have multiple values.

Example for Search:

```
@DOCUMENT { http://www.siroe.com/
    title{17}:   Welcome to Siroe!
    author{13}:  Dot Punchcard
}
```

A Search object has URL and schema-name fields to store its URL and `schema_name`:

```
char *url;              /* The URL */
char *schema_name;      /* The Schema-Name, such as @document or @RDMHeader*/
```

A Search object contains a collection of `SearchAVPair` objects, which each contain an attribute and one or more values. To access attribute values in a Search, use `Search_find()` to retrieve

the `AVPair` for the given attribute, or use `Search_findval()` to retrieve the value string for a given attribute. You must use all lowercase for attribute names for `find*()`, since only exact attribute name lookups are supported.

You can create Search objects by using the `Search_create()` function. You can also read Search objects from a Search stream.

`Search_Create`

```
NSAPI_PUBLIC Search *Search_Create
(char *schema_name, char *url)
```

Creates a Search structure with the given schema name and URL.

`Search_Free`

```
NSAPI_PUBLIC void Search_Free(Search *)
```

Frees the given Search structure.

`Search_GetTotalSize`

```
NSAPI_PUBLIC int Search_GetTotalSize(Search *s)
```

Gets the estimated total size of the Search in bytes.

`Search_GetAttributeCount`

```
NSAPI_PUBLIC int Search_GetAttributeCount(Search *s)
```

Gets the number of attributes in the Search.

`Search_GetAttributeSize`

```
NSAPI_PUBLIC int Search_GetAttributeSize(Search *s)
```

Gets the size of the attributes only.

`Search_GetValueSize`

```
NSAPI_PUBLIC int Search_GetValueSize(Search *s)
```

Gets the size of the values only.

`Search_GetValueCount`

```
NSAPI_PUBLIC int Search_GetValueCount(Search *s)
```

Gets the number of values only.

`Search_Merge`

```
NSAPI_PUBLIC int Search_Merge(Search *dst, Search *src);
```

Use this function to merge two Search objects (perform a Union of their attribute-values). It returns non-zero on error; otherwise, returns zero and the "`dst`' Search object contains all the attribute-value pairs from the "`src`' Search object.

If the "`dst`' object contains the same attribute as "`src`', then the attribute becomes a multi-valued attribute and all of the values are copied over to "`dst`'. Only multi-valued attributes are copied over. For single-value attributes, discard the value in "`dst`'. Currently only "`classification`" is a multi-valued attribute.

`Search_Find`

```
#define Search_Find(search, attribute-name)
```

Retrieves the `AVPair` for the given attribute in the given search. For example, the following statement gets the `AVPair` for the title attribute in the search `s`:

```
SearchAVpair avp=Search_Find(s, "title");
```

`Search_Findval`

```
#define Search_Findval(search, attribute-name)
```

Retrieves the value string for the given attribute in the given search. For example, the following statement prints the value of the title attribute of the search `s`:

```
printf("Title = %s\\n", Search_Findval(s, "title"));
```

`Search_Remove`

```
#define Search_Remove(search, attribute-name)
```

Removes the given attribute from the given search.

`Search_Insert`

```
#define Search_Insert(search, attribute-name,
value, value-size)
```

Inserts the given attribute and the value of the given size as an `AVPair` into the search.

`Search_InsertAVP`

```
#define Search_InsertAVP(search, avpair)
```

Inserts the given AVPair into the given search.

`Search_Apply`

```
#define Search_Apply(search, function, user-date)
```

Applies the given function with the given argument (user-data) to each `AVPair` in the given search. For example:

```
void print_av(Search *s, SearchAVPair *avp, void *unused)
{printf("%s = %s\\n", avp->attribute, avp->value);}

/* print every attribute and value in the search s */
Search_Apply(s, print_av, NULL);
```

Attribute-Value Pair Routines

Attribute-value pairs contain an attribute and an associated value. The value often is a simple null-terminated string; however, the value can also be binary data. Attribute-value pairs are stored as `SearchAVPair` structures.

The important fields in a `SearchAVPair` structure are:

`char *attribute;`	Attribute string; ”\\0’ terminated
`char *value;`	Primary value; may be ”\\0’ terminated
`size_t vsize;`	Number of bytes (8 bits) for primary value
`char **values;`	Multiple values for multivalued attributes
`size_t *vsizes;`	The sizes for the values
`int nvalues;`	Number of values associated with attribute
`int last_slot;`	Last valid slot - array may contain holes

`SearchAVPair_Create`	`NSAPI_PUBLIC SearchAVPair * SearchAVPair_Create (char *a, char *v, int vsz);` Creates an AVPair structure with the given attribute `a` and value `v`. The value `v` is a buffer of `vsz` bytes.
`SearchAVPair_Free`	`NSAPI_PUBLIC void SearchAVPair_Free(SearchAVPair *avp);` Frees the memory used by the given `SearchAVPair` structure
`Search_Replace`	`NSAPI_PUBLIC int Search_Replace(Search *s, char *att, char *val, int valsz);` Replaces the value of an existing attribute `att` with a new value val of size valsz in the Searches.
`Search_InsertStr`	`#define Search_InsertStr(search, attribute, value)` Inserts the given attribute with the given value into the search.
`Search_ReplaceStr`	`#define Search_ReplaceStr(search, attribute, value)` Replaces the existing value of the given attribute in the search with the given value.
`Search_Rename`	`NSAPI_PUBLIC int Search_Rename(Search *s, char *old_attr, char *new_attr);` Renames the given attribute to the given new name.
`Search_AttributeCompare`	`NSAPI_PUBLIC int Search_AttributeCompare (const char *a1, const char *a2);` Compares two attribute names. Returns 0 (zero) if they are equal, or non-zero if they are different. Case (upper and lower) and trailing `-s` are ignored when comparing attribute names. The following table illustrates the results of comparing some attribute names.

Multi-valued Attribute Routines

A Search attribute can have multiple values. Search supports the convention of using *-NNN* to indicate a multivalued attribute. For example, Title-1, Title-2, Title-3, and so on. The *-NNN* do not need to be sequential positive integers.

The Search Engine supports searching on multi-valued attributes such as the classification attribute. In Search representation, it is represented using classification-1, classification-2, and so on. For example:

```
classification-1{5}: robot
classification-2{5}: siroe
classification-3{10}: web crawler
```

`Search_AttributeCompareMV`	`NSAPI_PUBLIC int Search_AttributeCompareMV (const char *a1, const char *a2);` Compares two attribute names. Returns 0 (zero) if they are equal, or non-zero if they are different. If neither of the attributes is multi-valued then use above routine `Search_AttributeCompare()`. If one or both of the attributes are multi-value, use the base name of the multi-valued attribute for comparison. The base name of a multi-valued attribute is the name portion before "`-`". For example, the base name of classification-3 is classification.
`Search_MVAttributeParse`	`NSAPI_PUBLIC int Search_MVAttributeParse(char *a)` Returns the multi-valued number of the given attribute, and strips the attribute string of its `-NNN` indicator; otherwise, returns zero in the case of a normal attribute name. For example, classification-3 returns the number 3.
`Search_IsMVAttribute`	`NSAPI_PUBLIC char *Search_IsMVAttribute(const char *a` Returns NULL if the given attribute is not a multi-valued attribute; otherwise returns a pointer to where the multi-valued number occurs in the attribute string. For example, for the multi-valued attribute classification-3, it will return the pointer to 3.

`Search_InsertMV`	`NSAPI_PUBLIC int Search_InsertMV(Search *s, char *a, int slot, char *v, int vsz, int useval)`
	
	Inserts a new value `v` at index `slot` for the given attribute `a` (in non-multivalue form). If set, the `useval` flag tells the function to use the given value buffer rather than creating its own copy.
	
	For example:
	
	`Search_InsertMV(s, "classification", 3, "web crawler", strlen("web crawler");`
	
	Inserts
	
	`classification-3{10}: web crawler`

`Search_ReplaceMV`	`NSAPI_PUBLIC int Search_ReplaceMV(Search *s, char *a, int slot, char *v, int vsz, int useval);`

`Search_DeleteMV`	`NSAPI_PUBLIC int Search_DeleteMV (Search *s, char *a, int slot)`
	
	Deletes the value at the index slot in the attribute `a`. For example:
	
	`Search_DeleteMV(s, "classification", 3)`
	
	Deletes `classification-3`.

`Search_FindvalMV`	`NSAPI_PUBLIC const char *Search_FindvalMV (Search *s, const char *a, int slot)`
	
	Finds the value at the index slot in the attribute `a`. For example:
	
	`Search_FindvalMV(s, "classification", 3)`
	
	Returns web crawler (using the previous example).

`Search_SqueezeMV`	`NSAPI_PUBLIC void Search_SqueezeMV(Search *s)`
	
	Forces a renumbering to ensure that the multi-value indexes are sequentially increasing (for example, 1, 2, 3,...). This function can be used to fill in any holes that might have occurred `during Search_InsertMV()` invocations. For example, to insert values explicitly for the multivalue attribute author-*:
	
	`Search_InsertMV(s, "author", 1, "John", 4, 0);`
	`Search_InsertMV(s, "author", 2, "Kevin", 5, 0);`
	`Search_InsertMV(s, "author", 6, "Darren", 6, 0);`
	`Search_InsertMV(s, "author", 9, "Tommy", 5, 0);`
	`Search_FindvalMV(s, "author", 9); /* == "Tommy" */`
	`Search_SqueezeMV(s);`
	`Search_FindvalMV(s, "author", 9); /* == NULL */`
	`Search_FindvalMV(s, "author", 4); /* == "Tommy" */`

`SearchAVPair_IsMV`	`#define SearchAVPair_IsMV(avp)` Use this to determine if the `AVPair` has multiple values or not.
`SearchAVPair_NthValid`	`#define SearchAVPair_NthValid(avp,n)` Use this to determine if the Nth value is valid or not.

`SearchAVPair_NthValue`	`#define SearchAVPair_NthValue(avp,n)  ((avp)->values[n])`
	
	Use this to access the Nth value. For example:
	
	`for (i = 0; i <= avp->last_slot; i++)`
	`  if (SearchAVPair_NthValid(avp, i))`
	`    printf("%s = %s\\n", avp->attribute,`
	`      SearchAVPair_NthValue(avp, i));`

`SearchAVPair_NthVsize`	`#define SearchAVPair_NthVsize(avp,n) ((avp)->vsizes` Use this to get the size of the Nth value.
`Search_Contains`	`NSAPI_PUBLIC boolean_t Search_Contains (Search *s, char *a, char *v, int vsz);` Indicates if the given attribute contains the given value. It returns `B_TRUE` if the value matches one or more of the values of the attribute a in the given Searches.

Stream Routines for Parsing and Printing Searchs

A `SearchStream` contains one or more Search objects.

The general approach is that you use Search streams to create and process streams of many Search objects. Given a Search stream, you can parse it to get the Search objects from it. Use the `parse()` routine to get the next Search object in a Search stream. You can use `SearchStream_IsEOS()` to check whether the last object has been parsed.

You can use filtering functions for a Search stream to specify that certain Search attributes are allowed or denied. If an attribute is allowed, you can parse and print that attribute for Search objects in the stream. If it is denied, you cannot parse or print that attribute of Search objects in the stream.

Search streams can be disk or memory based.

When you create a `SearchStream`, you need to specify if you will be printing or parsing the Search stream, and if you will be using a memory- or disk-based stream. The functions you need to use will depend on what you will be doing with the Search stream.

For creating a Search streams into which you will be printing Searches, the functions are:

`Search_PrintInitFile()`	Creates a disk-based stream ready for printing.
`Search_PrintInitStr()`	Creates a memory-based stream ready for printing.
`Search_PrintInitFn()`	Creates a generic application-defined stream ready for printing. The given ”`write_fn`’ is used to print the stream.

To create Search stream from a file or a string containing Search, use the following functions:

`Search_ParseInitFile()`	Creates a disk-based stream ready for parsing. The stream is created from an input containing Search syntax.
`Search_ParseInitStr()`	Creates a memory-based stream ready for parsing. The stream is created from an input containing Search syntax.

SearchStream objects have a caller-data field, which you can use as you like:

```
void *caller_data;   /* hook to be used by caller */
```

Use `SearchStream_Parse()` to get the Search objects from the Search stream, and use `SearchStream_Print()` to write Search objects to the Search stream.

When you've finished with the stream, close it by using `SearchStream_Finish()`. Use `SearchStream_SetFinishFn()` to trigger the given `finish_fn` function.

The following example code takes a Search stream in `stdin` and prints each Search in the stream to `stdout`. Notice that this code uses `Search_ParseInitFile()` to create the SearchStream to parse the input file, and uses `Search_PrintInitFile()` to create the stream to print the Searches to `stdout`.

```
SearchStream *searchin = Search_ParseInitFile(stdin);
SearchStream *searchout = Search_PrintInitFile(stdout);
Search *s;
while (!SearchStream_IsEOS(searchin)) {
    if ((s = SearchStream_Parse(searchin)) {
        SearchStream_print(searchout, s);
        Search_Free(s);
    }
}
```

`Search_PrintInitFile`	`NSAPI_PUBLIC SearchStream *S earch_PrintInitFile(FILE *file)` Creates a disk-based stream ready for printing.
`Search_PrintInitStr`	`NSAPI_PUBLIC SearchStream * Search_PrintInitStr(SearchBuffer *memory)` Creates a memory-based stream ready for printing.
`Search_PrintInitFn`	`NSAPI_PUBLIC SearchStream *Search_PrintInitFn(int (*write_fn)(void *data,char *buf, int bufsz), void *data)` Creates a generic application-defined stream ready for printing. The given `write_fn` is used to print the stream. This function allows you to hook up your own routine for printing.
`Search_ParseInitFile`	`NSAPI_PUBLIC SearchStream *Search_ParseInitFile(FILE *fp)` Creates a disk-based stream ready for parsing. The file must contain Search-formatted data. The function reads Search data from the file object `fp`.

`Search_ParseInitStr`	`NSAPI_PUBLIC SearchStream * Search_ParseInitStr(char *buf, int bufsz)` Creates a memory-based stream ready for parsing. The character buffer must contain Search-formatted data.
`SearchStream_Finish`	`NSAPI_PUBLIC int SearchStream_Finish (SearchStream *)` Closes the stream when you have finished with it.
`SearchStream_SetFinishFn`	`NSAPI_PUBLIC int SearchStream_SetFinishFn (SearchStream *, int (*finish_fn)(SearchStream *))` Allows you to hook up a function for cleaning up after the Search stream finishes its business. The `finish_fn` will be called when `SearchStream_Finish()` has finished executing.
`SearchStream_Print`	`#define SearchStream_Print(ss, s)` Prints another Search object to the Search stream `ss`. Returns 0 on success, or non-zero on error.
`SearchStream_Parse`	`#define SearchStream_Parse(ss)` Parses and returns the next Search object in the Search stream.
`SearchStream_IsEOS`	`#define SearchStream_IsEOS(s)` Returns 1 (true) if the Search stream has been exhausted.
`SearchStream_IsPrinting`	`#define SearchStream_IsPrinting(s)` Returns 1 (true) if the Search has been set up in a stream by `Search_PrintInitFile()` or `Search_PrintInitStr()`.
`SearchStream_IsParsing`	`#define SearchStream_IsParsing(s)` Returns 1 (true) if the Search has been setup in a stream by `Search_ParseInitFile()` or `Search_ParseInitStr()`.

Filtering Search Objects

To support targeted parsing and printing, you can use the attribute filtering mechanisms in the Search stream. For each Search stream object, you can associate a list of allowed attributes. When printing a Search stream, only the attributes that match the allowed attributes will be printed. When parsing a Search stream, only the attributes that match the allowed attributes will be parsed.

`SearchStream_IsAllowed()` and `SearchStream_SetAllowed()` allow attributes, while `SearchStream_IsDenied()` and `SearchStream_SetDenied()` deny attributes. You can allow or deny an attribute, but not both.

`SearchStream_IsAllowed`	`NSAPI_PUBLIC boolean_t SearchStream_IsAllowed (SearchStream *ss, char *attribute);` Indicates that the given attribute is allowed (that is, it can be printed or parsed).
`SearchStream_SetAllowed`	`NSAPI_PUBLIC int SearchStream_SetAllowed (SearchStream *ss, char *allowed_attrs[])` Sets all the attributes in the `allowed_attrs` array to allowed.
`SearchStream_SetDenied`	`NSAPI_PUBLIC int SearchStream_SetDenied (SearchStream *ss, char *denied_attrs[]);` Sets all the attributes in the `allowed_attrs` array to be denied (that is, they cannot be parsed or printed).
`SearchStream_GetAllowed`	`NSAPI_PUBLIC char **SearchStream_GetAllowed (SearchStream *ss)` Returns an array of all the attributes that are allowed.
`SearchStream_GetDenied`	`NSAPI_PUBLIC char **SearchStream_GetDenied (SearchStream *ss);` Returns an array of all the attributes that are denied.

Memory Buffer Management

You can use Search buffers in parsing or printing routines. They take care of memory allocation for inserting and appending. They are basically memory blocks that are easy for Search routines to use.

A Search Buffer is represented in a `SearchBuffer` structure, that is created with the `SearchBuffer_Create()` function and freed with the `SearchBuffer-Free()` function. The `SearchBuffer` structure provides the `append()`, `increase()`, and `reset()` functions for manipulating the data in the buffer.

`SearchBuffer_Create`	`NSAPI_PUBLIC SearchBuffer * SearchBuffer_Create(int default_sz);` The `SearchBuffer` is used in `Search_PrintInitStr(SearchBuffer *memory)`. Before you can print Search to memory, you need to create a buffer for output.
`SearchBuffer_Free`	`NSAPI_PUBLIC void SearchBuffer_Free (SearchBuffer *sb);` Releases the memory buffer created by `SearchBuffer_Create()`.
`append`	`void (*append)(SearchBuffer *sb, char *data, int n)` Copies `n` bytes of data into the buffer.
`increase`	`void (*increase)(SearchBuffer *sb, int add_n)` Increases the size of the data buffer by `add_n` bytes.
`reset`	`void (*reset)(SearchBuffer *sb)` Resets the size of the data buffer and invalidates all currently valid data. A buffer can be reused by resetting it this way.

◆ ◆ ◆ CHAPTER 25

25 Overview of RDM

This chapter contains the following sections:

Introduction to RDM

A Resource Description Message (RDM) is a messaging format which two processes can use to exchange resource descriptions across a network. In RDM, one process (a client or agent) sends a request RDM message to a remote server which processes the request, then sends a response RDM message, similar to the `HTTP/1.0 request/response` model.

For more information about RDM, see:

`http://www.w3.org/TR/NOTE-rdm.html`

For example, you can send an RDM request to a search engine database to request RDs that match certain criteria. The search engine will send back an RDM response that contains the requested RDs (for example, all documents containing the string `style_sheets`).

RDM Format Syntax

Each RDM message contains a header and a body. The header identifies the nature of the RDM message, while the body contains any data required to carry out the needed request (for example, scoping criteria). Both the header and body of the RDM message is encoded using Search.

RDM Header

The RDM header section begins with a Search object whose schema name is `RDMHEADER` which must contain as least the attributes listed in the following table:

`rdm-version`
: A string identifying the version of the message specification. (for example, 1.0).

`rdm-type`
: A string identifying the nature of this message.

`rdm-query-language (required only for Request messages)`
: A string which identifies which query language is used in the given request.

`catalog-service-id (optional)`
: A `CSID` identifying the catalog to which the request/response applies (for example, `x-catalog://siroe.budgie.com:80/finance`. If not present, the RDM server will use its default catalog.

The following example shows two RDM headers and their attributes:

```
@RDMHEADER { -
    rdm-version{3}: 1.0
    rdm-type{14}: status-request
}

@RDMHEADER { -
    rdm-version{3}: 1.0
    rdm-type{10}: rd-request
    rdm-query-language{6}:  search
    catalog-service-ID{39}: x-catalog://budgie.siroe.com:80/finance
}
```

RDM Body

The RDM message body contains any data needed to carry out the request. For example, if the header is `rd-request`, which indicates a query request, the body must contain the query criteria or scope. For example, if you are sending a request to find all documents that contain the string `varrius`, then the body must be Search object whose schema-name is `RDMQuery` and whose scope attribute is `varrius`:

```
@RDMQUERY { -
    scope{7}:varrius
}
```

If the RDM message is a query request, the body of the message can also indicate which attributes of the resulting RDs are wanted, the number of results, and sort preference. For

example, the following search criteria in the RDM body specifies the `URL`, `title`, `author`, and `last-modified` attributes. The result set contains at most 10 hits and the resulting set is ordered by the `title` attribute:

```
@RDMQUERY { -
    scope{7}:varrius
    view-attributes{30}: url,title,author,last-modified
    view-hits{2}: 10
    view-order{5}: title
}
```

If the RDM message is another kind of request, such as a `Schema-Description-Request`, a `Server-Description-Request`, or a `Status-Request`, the body of the message should contain appropriate data. See the relevant document for more details on RDM bodies.

◆ ◆ ◆ CHAPTER 26

About the RDM API

This chapter

Introduction to the RDM API

This chapter describes the use of the search engine Resource Description Manager (RDM) API to access the search engine and its database. The RDM API provides routines to parse, modify, and create resource description messages (RDMs) using C.

The robot generates RDs and saves them to the search engine database. You can retrieve the RDs in Search format, use the RDM API to modify them, and then have the search engine import them back.

The RDM API is defined in the `rdm.h` file in the following directory in your search engine installation directory:

PortalServer-base`/sdk/rdm/include`

This chapter is restricted to discussing the use of C functions that come with the search engine RDM API. Therefore, it is strongly recommended that you have a basic understanding of the C programming language.

rdm.h File

The file rdm.h defines structures and functions for creating RDMs. The intent of these functions is to construct queries and instructions that can be output as RDM and sent via the sendrdm program to a search engine for processing. These queries and instructions are created in the C programming language.

The basic structures are:

- RDMHeader
- RDMQuery

To support each of these main structures, there are additional structures like RDMViewAttributes, RDMViewOrder, and RDMViewHit which are used to represent attributes in an RDMQuery.

The RDM API defined in rdm.h file provides functions for creating and modifying these structures. For example, the following statement:

```
RDMQuery *myquery = RDMQuery_Create("varrius");
```

creates an RDMQuery that corresponds to the following Search:

```
@RDMQUERY { -
    scope{7}:varrius
}
```

The following statement:

```
RDMQuery_SetViewAttr(myquery, "URL,Title");
```

modifies the RDMQuery so that it corresponds to the following Search:

```
@RDMQUERY { -
    scope{7}:varrius
    view-attributes{30}: URL,Title,Author,Last-Modified
}
```

Both the RDMHeader and RDMQuery structures have search fields, which contain the Search data for the header or the query. To extract Search data from RDMHeader or RDMQuery, you can access the search field. Thus, you can also use the RDM Search API to create and modify RDMHeader and RDMQuery objects.

API Reference

This section describes the RDM API, located in directory *PortalServer-base*/sdk/rdm/include/rdm.h.

Finding the RDM Version

RDM_Version

```
NSAPI_PUBLIC const char *RDM_Version(void);
```

Returns the version of the RDM library.

Creating and Freeing RDM Structures

For most RDMX structures, such as RDMRequest, RDMQuery, and so on, there are two or more creation functions. There is usually an RDMX_Parse() function, which takes Search arguments, and an RDMX_Create() function, which takes non-Search arguments.

There is also an RDMX_Free() function, which releases the object.

Several RDM structures also have an RDMX_merge() function, which merges data from a Search object into an existing RDMX structure.

An RDMHeader

An RDMHeader represents the header of an RDM. An RDMHeader structure has one public field, Search, which is a Search containing a collection of attribute-value pairs. The allowable attribute names of the attribute-value pairs in the Search are:

- rdm-version (required) - This is set automatically if you use RDMHeader_CreateRequest() or RDMHeader_CreateResponse() to create the RDMHeader.
- rdm-type (required).
- catalog-service-id (recommended) - This indicates which search engine instance to send the RDM request to).
- rdm-query-language (required for a request) - For communicating with the search engine, you can specify search.

Response RDMs also have several attributes, and you can find them in rdm.h.

The following statements create an RDMHeader whose RDMType is RDM_RD_REQ, and whose query language is search. This RDM will be sent to the search engine instance search1 of the search engine at http://budgie.siroe.com:6714.

```
CSID *csid = CSID_Parse("x-catalog://budgie.siroe.com:6714/search1");
RDMHeader *myheader = RDMHeader_CreateRequest(RDM_RD_REQ, "search", csid);
```

The allowable values for RDM-Type and their description are:

RDM_MSGTYPE_UNKNOWN	Undefined or unknown message type
RDM_RD_UPD_REQ	Requesting an RD update
RDM_RD_REQ	Requesting an RD update (same as RDM_RD_UPD_REQ)
RDM_RD_DEL_REQ	Requesting an RD delete
RDM_RD_UPD_RES	Responding to an RD update request
RDM_RD_RES	Responding to an RD update request (same as RDM_RD_UPD_RES)
RDM_RD_DEL_RES	Responding to an RD delete request
RDM_SD_REQ	Requesting a server description
RDM_SD_RES	Responding to a server description request
RDM_SCH_REQ	Requesting a schema description
RDM_SCH_RES	Responding to a schema description request
RDM_TAX_REQ	Requesting a taxonomy description
RDM_TAX_RES	Responding to a taxonomy description request
RDM_STAT_REQ	Requesting a status
RDM_STAT_RES	Responding to a status description request

You can use the following macros to get and set the string values of the attributes:

```
RDMHeader_GetType(RDMHeader) /* returns RDMType */
RDMHeader_GetVersion(RDMHeader)
RDMHeader_GetQueryLanguage(RDMHeader)
RDMHeader_GetCSID(RDMHeader)
RDMHeader_GetResponseInterpret(RDMHeader)
RDMHeader_GetErrorMessage(RDMHeader)
RDMHeader_GetErrorNumber(RDMHeader)

RDMHeader_SetType(RDMHeader,RDMType)
RDMHeader_SetVersion(RDMHeader,stringvalue)
RDMHeader_SetQueryLanguage(RDMHeader,stringvalue)
RDMHeader_SetCSID(RDMHeader,stringvalue)
```

```
RDMHeader_SetResponseInterpret(RDMHeader,stringvalue)
RDMHeader_SetErrorMessage(RDMHeader,stringvalue)
RDMHeader_SetErrorNumber(RDMHeader,stringvalue)
```

RDMHeader_Parse	NSAPI_PUBLIC RDMHeader *RDMHeader_Parse (SearchStream *ss);
RDMHeader_Create	NSAPI_PUBLIC RDMHeader *RDMHeader_Create(RDMType t);
RDMHeader_CreateRequest	NSAPI_PUBLIC RDMHeader *RDMHeader_CreateRequest (RDMType, char *ql, CSID *)
RDMHeader_CreateResponse	NSAPI_PUBLIC RDMHeader *RDMHeader_CreateResponse (RDMType, char *ri, CSID *);
RDMHeader_Free	NSAPI_PUBLIC int RDMHeader_Free(RDMHeader *r);
RDMHeader_Merge	NSAPI_PUBLIC int RDMHeader_Merge(RDMHeader *, Search Merges data from a Search object into the RDMHeader object.

An RDMQuery

An RDMQuery represents the body of an RDM. An RDMQuery structure has one public field, search, which is a Search containing a collection of attribute-value pairs. The allowable attribute names of the attribute-value pairs in the Search are:

- scope (required)
- view-attributes (optional)
- view-hit (optional)
- view-order (optional)
- view-template (optional)

You can use the following macros to get and set the string values of these attributes:

```
RDMQuery_GetScope(query)
RDMQuery_GetViewAttr(query)
RDMQuery_GetViewHits(query)
RDMQuery_GetViewOrder(query)
RDMQuery_GetViewTemplate(query)

RDMQuery_SetScope(query, value)
RDMQuery_SetViewAttr(query,value-list)
RDMQuery_SetViewHits(query,value)
RDMQuery_SetViewOrder(query,value-list)
RDMQuery_SetViewTemplate(query,value)
```

RDMQuery_Parse	NSAPI_PUBLIC RDMQuery *RDMQuery_Parse(SearchStream *ss);
RDMQuery_Create	NSAPI_PUBLIC RDMQuery *RDMQuery_Create(const char *scope);
RDMQuery_Free	NSAPI_PUBLIC int RDMQuery_Free(RDMQuery *);
RDMQuery_Merge	NSAPI_PUBLIC int RDMQuery_Merge(RDMQuery *, Search *);

Other RDM Structures

In addition to RDMHeader and RDMQuery, the file rdm.h provides definitions and functions for the following auxiliary objects. See the rdm.h file in *PortalServer-base*/sdk/rdm/include directory for details.

- RDMRequest
- RDMResponse
- RDMServer
- RDMView
- RDMViewHits
- RDMViewOrder
- RDMTaxonomy
- RDMClassification
- RDMSchema

◆ ◆ ◆ CHAPTER 27

Example of Submitting a Query

This chapter contains the following sections:

Overview of Submitting a Query

To send the `RDMHeader` and `RDMQuery` objects to the search engine to perform a query request, you can use the pre-built `sendrdm` program (located in *PortalServer-base*`/lib` directory). This program takes an RDM request (which is a message in Search format), sends it to the search engine, and outputs the results as a Search stream. You can also use HTTP to post an RDM stream directly to the server through its URL `http://`*server*`:`*port*`/`*search-ID*`/search`). The program must be run in a search-enabled Sun Java System Portal Server software instance directory such as the default *PortalServer-DataDir*`/searchservers/search1/` directory.

You should also change the definitions for `MY_CSID`, `MY_SCOPE` and `MY_ATTR_VIEW` to suit your needs.

RDMHeader and RDMQuery Object Parameters

The list of `RDMHeader` and `RDMQuery` object parameters and their description are:

`MY_CSID` — ID of the search engine instance. This parameter specifies the specific search engine that you created in the search engine Administration Interface. You can find the exact value in `/var/opt/SUNWportal/searchservers/search1/config/search.conf`. For SSL enabled server instances, use x-catalogs instead of x-catalog in the CSID.

`MY_SCOPE`	Query string. The default is to search for documents containing the string `varrius`. For example, if you want to search for all documents containing the string `style_sheets`, then set `MY_SCOPE` to `style_sheets`.
`MY_ATTR_VIEW`	Attributes to be printed, such as URL, title and description.

EXAMPLE 27–1 RDM API to Submit a Query Example

This example generates an RDM for querying a search engine database. You can pipe the output of the sample program to a temporary file and then use the `sendrdm` program to post it to the search engine.

```
/******* Example Use of the RDM API to submit a query *******/

#include <stdio.h>
#include “Search.h”
#include “rdm.h”

#define MY_SCOPE “varrius”
#define MY_CSID “x-catalog://budgie.siroe.com:6714/search1”
#define MY_ATTR_VIEW “title,content-type”

int main(int argc, char *argv)
{
    RDMQuery *myquery = NULL;
    CSID *csid = NULL;
    RDMHeader *myheader = NULL;
    SearchStream *out = Search_PrintInitFile(stdout);

    /* Create the RDMQuery and specify its scope */
    if(!(myquery = RDMQuery_Create(MY_SCOPE))) {
        perror(“RDMQuery_Create\\n”);
        exit(-1);
    }

    /* Set the view attributes of the RDMQuery */
    RDMQuery_SetViewAttr(myquery, MY_ATTR_VIEW);

    /* Create the CSID that points to your search engine instance */
    if(!(csid = CSID_Parse(MY_CSID))) {
        perror(“CSID_Parse\\n”);
        exit(-1);
    }

    /* Create the RDMHeader */
    if(!(myheader = RDMHeader_CreateRequest(RDM_RD_REQ, “search”,
    csid))) {
```

EXAMPLE 27–1 RDM API to Submit a Query Example *(Continued)*

```
        perror("RDMHeader_CreateRequest\\n");
        exit(-1);
    }

    /* print the RDMHeader to the output Search stream */
    /* print the RDMQuery to the output Search stream */
    (*out->print)(out, myheader->search);
    (*out->print)(out, myquery->search);

    /* free the structures and exit */
    RDMHeader_Free(myheader);

    RDMQuery_Free(myquery);
    SearchStream_Finish(out);
    exit(0);
}
/********** EOF ***************/
```

Running the Example

To run the example described in Example 27–1, follow these steps.

▼ To Run the Example

1. **Edit the definitions for `MY_SCOPE`, `MY_CSID`, and `MY_ATTR_VIEW` as appropriate for your situation. For more information, see "Running the Example" on page 267.**

2. **Save the file in *PortalServer-base*`/sdk/rdm/examples` directory. For example, save the file as `example4.c` in *PortalServer-base*`/sdk/rdm/examples` directory.**

3. **Create a makefile. You can find sample makefiles at *PortalServer-base*`/sdk/rdm/examples` directory. Edit the makefile to include `example4.c`. Following is a makefile with the changes needed for `example4.c` in bold:**

   Makefile for Search/RDM examples

```
#  Makefile for Search/RDM SDK examples
#  Use make and cc.
CC              = cc
SDKDIR          = ..
SDKLIB          = $(SDKDIR)/lib/librdm.a
SDKINC          = $(SDKDIR)/include/
```

```
CFLAGS          = -I$(SDKINC) -DXP_UNIX
CFLAGS          += -DSOLARIS
#CFLAGS         += -DIRIX
#CFLAGS         += -DHPUX
#CFLAGS         += -DAIX
EXAMPLES        = example1 example2 example3
                  example4
all:    $(EXAMPLES)
example1:example1.o
$(CC) -o $@ $@.o $(SDKLIB)
example2:example2.o
$(CC) -o $@ $@.o $(SDKLIB)
example3:example3.o
$(CC) -o $@ $@.o $(SDKLIB)
example4:example4.o
$(CC) -o $@ $@.o $(SDKLIB)
```

4 **From the** *PortalServer-base*`/sdk/rdm/examples` **directory, build the example as follows:**

```
Solaris:gmake
```

5 **From the** *PortalServer-base*`/sdk/rdm/examples` **directory, run the** `example4.c` **program to generate the RDM file.**

```
example4.c > rdm.search
```

Following is an example of `rdm.search` file.

```
@RDMHEADER { -
 catalog-service-id{40}:x-catalog://budgie.siroe.com:6714/budgie
 rdm-version{3}: 1.0
 rdm-type{10}: rd-request
 rdm-query-language{6}: search
}

@RDMQUERY { -
 view-attributes{18}: title,content-type
 scope{5}: varrius
}
```

The file `rdm.search` created in must be placed into the server instance directory.

6 **Send the Search contained in** `rdm.search` **to the program** `sendrdm`**. For example, type:**

```
./run-cs-cli sendrdm -u /portal/search rdm.search
```

The current directory should be the server instance directory. If `rdm.search` is not in the server instance directory, reference the file from where is was created. For example:

a. **Change directories to** `server_instace_dir`**.**

b. Type `./run-cs-cli sendrdm -u /portal/search sdk_dir/rdm.search`.

c. The results of the `sendrdm` program will be a Search stream containing the results of the query, such as the following example:

```
@RDMHEADER { - catalog-service-id{41}:x-catalog://budgie.siroe.com:6714/
budgie rdm-version{3}: 1.0 rdm-type{11}: rd-response rdm-response-interpret{51}:
20 results out of 36281 hits across 88985 documents } @DOCUMENT
{ http://fury.sesta.com:999/it/newsref/pr/newsrelease417.html content-type{9}:
text/html score{3}: 100 title{17}: Comunicato stampa } @DOCUMENT
{ http://fury.sesta.com:999/it/newsref/pr/newsrelease374.html content-type{9}:
text/html score{3}: 100 title{17}: Comunicato stampa }
```

You can pipe the output of `sendrdm` (which is a `SearchStream`) to another program to print the results of the query.

◆ ◆ ◆ CHAPTER 28

Overview of the Search Engine Java SDK

This chapter contains the following sections:

Introduction to Search Engine Java SDK

To submit queries and add entries to the search engine database by using the Java programming language, you need to use the search engine Java Software Development Kit (SDK), which is available at:

PortalServer-base`/sdk/search`

The Java SDK includes the Java classes needed for interacting with the search engine database and a sample search application which can be run as an applet or as a stand alone program.

The Search Engine Java SDK

The classes in the search engine Java SDK provide a Java interface for interacting with the search server to access the search engine database.

Using the Java SDK, you can build Java programs, applets, and your own interfaces that submit searches to the search engine.

The search engine database contains resource descriptions for the indexed resources, such as documents. Each resource description is described in Search format, where Search stands for Summary Object Interchange Format.

For a discussion of Search, see:

```
http://www.w3.org/TR/NOTE-rdm.html#search.
```

Running the Sample Applications

The search engine Java SDK contains sample Java and HTML files. `SearchDemo.java` is an example Java program which can be run as an applet or as a stand alone program. To run the applet version, use the JDK applet viewer or a browser with the Java 1.2 plug-in. You must also edit `SearchDemo.html` to pass the information about your search engine to the SearchDemo applet. The command line version takes the server information as a command line argument

▼ To Install and Run the Search Demo Command Line Program

1. **Compile the `SearchDemo.java` file. Make sure the class path includes the SDK JAR file, `searchsdk.jar`. For example, type:**

   ```
   javac -classpath PortalServer-base/sdk/searchsdk.jar SearchDemo.java
   ```

2. **Create a directory hierarchy that reflects the Java package structure for these classes to use the compiled classes directly, and move the classes into the demo package of this hierarchy.**

   For example, type:

   ```
   mkdir -p com/sun/portal/search/demo; cp *.class com/sun/portal/search/demo
   ```

3. **Invoke the search demo from the command line and supply the address of the search server and the query string as arguments.**

   For example, type:

   ```
   java -classpath PortalServer-base
   /sdk/searchsdk.jar com.sun.portal.search.demo.SearchDemo http://
   portal_server_host_name:port/search-ID/search 'search query'
   ```

▼ To Install and Run the Search Demo Applet

To run the search demo as an applet, use the JDK applet viewer or a browser with the Java 1.3.1 plug-in. You must also edit `SearchDemo.html` to configure the information about your search engine location.

Note – The instructions included here are for applet viewer. To run the applet in a browser, you need Java 1.2 or later browser plugin. More information on the Java browser plug-in can be found at `http://java.sun.com/products/plugin` and `http://java.sun.com/products/plugin/1.3/docs/tags.html`.

1. **Compile the `SearchDemo.java` file. Make sure the class path includes the SDK JAR file, `searchsdk.jar`.**

   For example, enter:

   ```
   javac -classpath PortalServer-base/sdk/searchsdk.jar SearchDemo.java
   ```

2. **Create a directory hierarchy that reflects the Java package structure for these classes to use the compiled classes directly, and move the classes into the demo package of this hierarchy.**

   For example, type:

   ```
   mkdir -p com/sun/portal/search/demo; cp *.class com/sun/portal/search/demo
   ```

3. **Enter the server location information by editing the applet parameters in `SearchDemo.html`.**

   For example, type:

   ```
   <param name="RDMServer" value="http://portal_server_host_name:port/portal/search">
   ```

   Appletviewer can now be run from the command line. It will access the java classes from the current location. For example, enter:

   ```
   appletviewer SearchDemo.html
   ```

   Search queries are entered into the search bow that appears and results are sent to `stdout` of the controlling terminal. If you are running the applet in a browser, the results are displayed in the Java console.

   If you have installed the browser Java plugin, then the instructions are similar to those for appletviewer except that you must make the classes available for download to the client browser. To do this:

4. **Copy `SearchDemo.html`, `searchsdk.jar` and the directory hierarchy created above (including the SearchDemo classes) to a location that is in the content path of your web server.**

   You can also add the search demo classes to `searchsdk.jar` to make the applet available in a single download.

5. **The `searchsdk.jar` file must be named as the archive attribute of the applet tag in `SearchDemo.html`.**

   For example, enter:

   ```
   jar uf com/sun/portal/search/demo/*.class searchsdk.jar
   ```

◆ ◆ ◆ CHAPTER 29

Using Java To Access the Search Server Database

You can use the search engine Java SDK to write Java programs that interface with the `sendrdm` program to retrieve information from the search engine database.

The chapter contains the following sections:

Creating a Search Object

The entry point for submitting searches is the Search class. You need to create a new Search object, then call `doQuery()` on it to execute the search query.

The first thing you need to do is create a new Search object. The full constructor syntax is:

```
public Search(
    String scope
    String viewAttributes,
    String viewOrder,
    int firstHit,
    int viewHits,
    String queryLanguage,
    String database,
    String RDMServer,
    String ssoToken,
)
```

The arguments for the constructor and their description are:

`String scope` The query string or scope, that is, the string being searched for.

`String viewAttributes`	A comma-delimited list of the Search attributes to be retrieved from the database, such as URL, Author, Description. For example, `score,url,title,description,classification`.
`String viewOrder`	The order by which to sort the results. This is a comma-delimited list of attributes. Use the minus sign to indicate descending order, and a plus sign to indicate ascending order. For example, `-score,+title`.
`int firstHits`	The hit number of the first result to return. A typical value is 1.
`int viewHits`	Maximum number of results to return. A typical value is 10.
`String queryLanguage`	The Search Server query language. You should use search for a normal query.
`String database`	The logical name of a database (or collection) you wish to search. A typical value is null which will search the server's default database.

`String RDMServer`	The URL of the search engine servlet. This argument has the form:
	
	`http://`*hostname.domain.com*`:`*port*`/`*portal-ID*`/search`
	
	For example, if the search server is installed on `www.yourcompany.com` on port 80, the value would be:
	
	`http://www.`*yourcompany*`.com:80/`*portal-ID*`/search`

`String ssoToken`	An Sun Java System Access Manager software single sign on token used when doing secure searches. There is also a simpler convenience constructor with the following syntax: `public Search(String scope, String RDMServer)`

When this constructor is used, the following values are used for the unspecified arguments:

```
viewAttributes: null. Return all attributes.
viewOrder: null. Use the server default sort order - sorted by relevance.
firsthit: 1. Start hits at hit number 1.
viewhits: 10. Return 10 hits only.
query language: search. Search for documents using the normal query language.
database: null. Search the server's default database.
ssoToken: null. Use anonymous search.
```

Executing A Query and Getting the Results

You submit a query by calling the `doQuery()` method.

```
public void doQuery()
```

The results from `Search.doQuery()` can be obtained as a Search stream using `Search.getResultStream()`. The next search will replace the previous result stream reference, so you must process the results or save a reference to the result stream after each query. There are also methods for checking the number of results.

```
public SearchInputStream getResultStream()
```

The function `getResultStream()` returns a `SearchInputStream` which is used to read the Search hit objects. Each Search object read from the stream corresponds to one result.

```
public int getHitCount()
```

The function `Search.getHitCount()` returns the number of hits that matched the query.

```
public int getResultCount()
```

The function `Search.getResultCount()` returns the number of results that were returned by the server. The result count will be equal to the number requested by the `viewHits` argument whenever there are enough results available.

```
public int getDocumentCount()
```

The function `Search.getDocumentCount()` returns the total number of documents searched across. This will usually equal the total number of documents in the searched database.

Working Through An Example Search Application

This section discusses the SearchDemo example application provided with the Java search SDK. The purpose of this example is to show how to use a Search object to submit a query to the search server and how to extract the results from the Search object. The example application is very simple, and limits use of Java to achieving the goals of the example. It creates a Java applet that presents the user with a text field in which to enter a search query, and a Search button to initiate the search. The results of the query are read from a `SearchInputStream` returned by the Search object. The query results are displayed to standard output as plain text.

Import the Necessary Classes

In your favorite editor or Java development environment, view the search SDK file `SearchDemo.java`. This demo runs as a stand alone application or as an applet. The `search` package is provided as part of the Search Server Java SDK, while `java.applet`, `java.awt`, and `java.io` are standard Java packages.

```
package com.sun.portal.search.demo;
import com.sun.portal.search.search.*;
import java.applet.Applet;
import java.awt.*;
import java.io.*;
```

Define the SearchDemo Class

The class SearchDemo is an applet, so it extends the class Applet. SearchDemo defines `init()` and `main()` methods which allow it to run as an applet or as a stand alone (command line) program.

EXAMPLE 29–1 SearchDemo

```
/**
* Applet/application for simple query interface. Can be used as an
* example for those who want to create their own java interface.
* This example demonstrates search only.  Browse, determining
* the schema of the search server and obtaining the taxonomy
* of the search server will be demonstrated in other examples.
*/

public class SearchDemo extends Applet {
    /** Run as an applet. */
    public void init() {
        String rdm = getParameter("RDMServer");
        SimpleSearch ss = new SimpleSearch(rdm);
        SearchPanel sp = new SearchPanel(ss);
        setLayout(new FlowLayout(FlowLayout.CENTER));
        add(sp);
    }
    /** Run as an application. */
    public static void main(String argv[]) throws Exception {
        int args = argv.length;
        String SearchOutputFile = null;
        if (args != 1 && args != 2 && args != 3) {
            System.out.println("args: RDMServer [query]
            [search_output_file_name]");
```

EXAMPLE 29–1 SearchDemo *(Continued)*

```
            return;
        }
        String rdm = argv[0];  // rdm search server, eg,
        // http://portal.siroe.com:2222/ps/search
        SimpleSearch ss = new SimpleSearch(rdm);
        if (args == 3) {
            --args;
            ss.setSearchfile(argv[2]); // dump raw Search results to this file
        }
        if (args == 1) {
            // run from a search box
            Frame f = new Frame();
            SearchPanel sp = new SearchPanel(ss);
            f.add(sp);
            f.pack();
            f.show();
        }
        else {
            // run from command line
            String query = argv[1];
            ss.doSearch(query);
        }
    }
}
```

There is a helper class called SearchPanel which handle the applet GUI. It sets up a search panel with a text box to enter a query and a submit button to run the query. See the source file for more details.

Define the SimpleSearch Class

Notice the private helper class SimpleSearch. This is where the search is set up and executed and we will look at it in more detail here. The applet/command line class SearchDemo sets up the arguments for SimpleSearch using either applet or command line parameters. It then calls the `SimpleSearch.doSearch(String scope)` method to execute the search and display the results. The SimpleSearch constructor takes the location of the search server as an argument. In this way, a single SimpleSearch object can be used repeatedly to run searches against the remote search server.

The `SimpleSearch.setSearchfile(String filename)` method is used by the main program to direct search results to a file when running in command line mode.

EXAMPLE 29–2 SimpleSearch Class

```
/** Performs a simple search and displays its results. */
class SimpleSearch {
    String RDMServer;
    String SearchOutputFile;
    /**
    * SimpleSearch constructor
    * @param rdm - the rdm search server,
        eg, http://portal.siroe.com:2222/portal/search
    */
    public SimpleSearch(String rdm) {
        System.out.println("Sun Java System Search Java Demo");
        RDMServer = rdm;
    }
    /**
    * @param filename - a file to dump raw Search results into - only
    * use if running from the comand line or an applet with file
    * system access
    */
    public void setSearchfile(String filename) {
        SearchOutputFile = filename;
    }
    /** Execute a search */
    public void doSearch(String scope) throws IOException {
        ...see
```

"Define the SimpleSearch Class" on page 279

```
...
    }
}
```

Before submitting the search, SimpleSearch needs to create a Search object. The constructor for the Search class takes several arguments as discussed previously.

EXAMPLE 29–3 doSearch Class

```
/** Execute a search */
public void doSearch(String scope) throws IOException {
    /* The Search class encapsulates the search.
    ** It's parameters are:
    **  1) the search string
    **  2) the attributes you want returned, comma delimited
    **  3) sort order, comma delimited, - descending, + ascending
    **  4) first hit
    **  5) number of hits
```

EXAMPLE 29–3 doSearch Class *(Continued)*

```
    **  6) query language, eg search, taxonomy-basic, schema-basic, etc
    **  7) database to search
    **  8) The RDM server URL, eg, http://portal.siroe.com:2222/ps/search
    **  9) Access token (null for anonymous access, or valid iPlanet
                Directory Server Access Management Edition session id)
    */
    Search  search = new Search(
        scope,
        "score,url,title,description",
        "-score",
        1,
        20,
        "search",
        null,
        RDMServer,
        null
    );
```

The Search constructor arguments used here are:

`scope`
: The search scope is the actual query run by the search server. It is the scope argument to `doSearch()` and ultimately derives from either the applet input panel or a command line argument to the main program.

`viewAttributes = "score,url,title,description"`
: The requested attribute set shown here will result in the server returning the score, url, title, and description of all documents that match the query.

`viewOrder = "-score"`
: A comma delimited list of the attributes to be used to sort the results. A minus sign indicates descending order, a plus sign indicates ascending order. In this case, sort the results by decreasing numerical score value, and use alphabetical order of the title as the secondary sort order.

`firstHit = 1`
: The hit number of the first returned result.

`viewHits = 20`
: The maximum number of results to return.

`queryLanguage = "search"`
: The search server query language. Use search for normal searches.

`RDMServer`
The URL of the remote search engine, specified as an argument to the SimpleSearch constructor.

`ssoToken = null`
The Access Manager software single sign on token. Not used in this case, implying anonymous search.

Execute the Search Query

An output stream is created to hold the search results and paginate through the search results for a fixed number of pages, in this case five pages in total, where each page has `viewHits` (=20) results. The first page starts with the first hit (firstHit=1). The search is executed again for each page of results. It is possible to cache the results for all pages with a single search of course, but it is often easier to simply resubmit the search each time. This is equivalent to a user clicking a next button in a search user interface.

```
/* Execute the query. */
System.out.println("\\nSearch results for '" + scope + "'");
 DataOutputStream sos = null;
if (SearchOutputFile != null) {
try {
sos = new DataOutputStream(new FileOutputStream(SearchOutputFile));
}
catch (Exception e1) {
System.out.println("Error: failed to create output file: " + e1);
}
}
int  pagenum = 1;
int  pagesize = 10;
SearchBuffer firstPageSearch = new SearchBuffer();
for (; pagenum <= 5; pagenum++) {
int firstHit = (pagenum-1)*pagesize+1;
try {
search.doQuery(firstHit, pagesize);
}
catch (Exception ex) {
ex.printStackTrace();
break;
}
// Check the result count. -1 indicates an error.
if (search.getResultCount() <= 0)
break;
```

The results are stored in the Search object. Now do something with the results. The functions `doSomethingWithResults()` and `displayHTMLResults()` will be defined in this file. They each show a different way of extracting the results from the Search object.

Display the Results

The example application displays the query results to standard output or to a named file. In reality, you would do more with the results than just print them like this, but once you know how to get the results out of the Search object, it is up to you what you do with them. You can use standard Java functionality to process the results in any way you like.

The Search object has a method called `getResultStream()` that returns a SearchInputStream object. Each result is read from this Search stream in turn. Note that the client server connection uses an efficient streamed protocol; it is conceivable that the server is still returning later results while the client is processing the first results. For each Search object read from the result stream you can use the `getValue()` method to get the value of a particular field, for example, `getValue("title")` gets the title of a Search object.

First, print out some general result information:

```
System.out.println("========================================");
System.out.println("page " + pagenum
+ ": hits " + search.getFirstHit()
+ " to " + (search.getFirstHit() + search.getResultCount() - 1)
+ " out of " + search.getHitCount()
+ " across " + search.getDocumentCount() + " documents");
System.out.println("========================================");
System.out.println();
```

Now, retrieve each search hit from the result stream as Search objects and print its URL, title, description, and score to the output stream (either the Java console, standard output, or a named output file).

```
SearchInputStream resultStream = search.getResultStream();
Search search;
/* Examine the results of the search. The following
* code loops through the stream of Search instances. */
for (search = resultStream.readSearch(); search != null;
search = resultStream.readSearch()) {
    // For illustration, dump out the entire Search on
            the first page only.
    if (pagenum == 1)
        firstPageSearch.write(search.toByteArray());
        /* Now we use the getValue() method to get
        * the values of each of the requested
        * attributes. URL is special and has
        * its own accessor method.
        */
        String  u = search.getURL();
        String  t   = search.getValue("title");
        String  d   = search.getValue("description");
```

```
        String  sc  = search.getValue("score");

        /* do something with the results */
        System.out.println(
            "TITLE:       " + t  + "\\n" +
            "URL:         " + u  + "\\n" +
            "SCORE:       " + sc + "\\n" +
            "DESCRIPTION: " + d  + "\\n" +
            "-----------------------------------------\\n"
        );
        // If there is a Search output file, write the Search
                data there too...
        if (sos != null) {
            try {
                sos.writeBytes(search.toString());
            }
            catch (Exception e1) {
                System.out.println("Error: failed to write to Search
                output file: " + e1);
            }
        }
    }
    // Break if the largest requested hit has been displayed
    if (search.getHitCount() <= (firstHit + pagesize - 1))
        break;
    }
    if (firstPageSearch == null)
        System.out.println("No matching documents found.");
    }
```

◆ ◆ ◆ CHAPTER 30

Using Java To Add Entries to the Search Engine Database

The program `rdmgr` is used to add data to the database from the command line. This chapter describes how to create input data for `rdmgr` so that it can be added to the database.

`rdmgr` Command

The `rdmgr` utility can add new data as well as replace, modify, or retrieve existing data. All data input and output is done using Search, with UTF-8 character encoding for character fields. Note that Search also supports binary-valued fields and they can be added or retrieved too.

For more information on `rdmgr`, see Technical Reference Guide.

In the simplest case, `rdmgr` can be used to add a file containing multiple Search Resource Descriptions (RDs) to the database. This is as simple as creating a Search file with the search sdk, or by other means, and adding the data with the command `rdmgr search_input_file`. The `rdmgr` also accepts resource description submit requests as input. Submit requests also use Search format and include a request header in addition to the normal body consisting of the Search data to be added or retrieved to or from the database.

Search Object

A Search object consists of a schema name (such as REQUEST or DOCUMENT), a URL, and a list of attribute-value pairs. The `com.sun.portal.search.search` package in the Search Server Java SDK is used to build Search objects and write them to a file.

Constructing and Submitting a Request

You can use the Search classes to create a RD submit request for input to `rdmgr`.

Constructing a Request

Here is an example of constructing a submit request that can be used as input to `rdmgr`. Request headers do not have an associated URL and use `"-"` instead.

```
Search req = new Search("REQUEST", "-");
```

A submit request can have the following attributes:

```
submit-csid
submit-database
submit-type
submit-operation
submit-view
```

Add values for each of these attributes to the request header. This example shows an update operation into the default database. The database attribute is optional, the default database is used if none is supplied. The submit view restricts which attributes are updated, by default all of the supplied input attributes will be updated for each resource description.

```
req.insert("submit-database", "default");
req.insert("submit-type", "nonpersistent");
req.insert("submit-operation", "update");
req.insert("submit-view", "title,author,description");
```

Now we create the body part of the submit request. We'll be updating the resource description of a document, whose URL is `http://www.sesta.com/~jocelyn/resdogs.index.htm`, whose title is "Saving English Springer Spaniels," whose author is Jocelyn Becker, and whose description is "English Springer Spaniels in need of homes."

```
Search data = new Search("DOCUMENT",
"http://www.sesta.com/~jocelyn/resdogs.index.htm\\n");
data.insert("title", "Saving English Springer Spaniels");
data.insert("author", "Jocelyn Becker");
data.insert("description", "English Springer Spaniels in need of homes");
```

Now, the request is saved to a file for input to `rdmgr`:

```
SearchOutputStream sos = new SearchOutputStream("search_file");
sos.write(req);
sos.write(data);
sos.close();
```

At this point `search_file` should contain:

```
@REQUEST { -
    submit-database{7}: default
    submit-type{13}: nonpersistent
    submit-operation{6}: update
    submit-view{24}: title,author,description
}
@DOCUMENT { http://www.best.com/~jocelyn/resdogs/index.html
    title{32}: Saving English Springer Spaniels
    author{14}: Jocelyn Becker
    description{42}: English Springer Spaniels in need of homes
}
```

Submitting a Request

When this input is processed by `rdmgr`, it will result in the attributes of the RD shown being updated to the database and indexed. The `rdmgr` utility supports other types of requests too.

EXAMPLE 30–1 `rdmgr` Submit

```
// submit header fields
String SUBMIT_CSID = "submit-csid";
String SUBMIT_TYPE = "submit-type";
String SUBMIT_OPER = "submit-operation";
String SUBMIT_VIEW = "submit-view";
String SUBMIT_DB = "submit-database";
String SUBMIT_MESSAGE = "message";
String SUBMIT_ERROR = "error";
// submit operations
String SUBMIT_RETRIEVE = "retrieve";
String SUBMIT_INSERT = "insert";
String SUBMIT_DELETE = "delete";
String SUBMIT_UPDATE = "update";
// submit types
String SUBMIT_PERSISTENT = "persistent";
String SUBMIT_NONPERSISTENT = "nonpersistent";
String SUBMIT_MERGED = "merged";
```

Submit Operations

The submit operations are as follows:

`retrieve` Retrieves the requested fields (the submit view) for the requested RDs. In this case the data is a list of RDs that can be specified by their URLs only. The server will return the requested fields for these RDs.

`insert` The server adds or replaces the RDs supplied to the database.

`delete` The server deletes the RDs. As with retrieve, it is sufficient to list the RDs by URL alone, it is not necessary to supply values for the fields of the RDs.

`update` The server modifies the RDs in the database by merging any existing fields with the fields supplied in the data. If an attribute view list is supplied, only those attributes will be updated. If a view is not supplied, all of the given input attributes will be updated for each RD.

Submit Types

The submit types are as follow:

`persistent` The operation is applied to the persistent part of each RD in the database. When an RD is retrieved from the database, or indexed, any persistent fields take precedence over non persistent fields. This allows you to manually edit the fields of an RD without having to worry that your edits will be lost the next time the RD is submitted by the robot, for example.

`non-persistent` This is the default type. Data is normally added as non-persistent data. Note that RDs are only indexed and searchable if they have a non-empty non-persistent component.

`merged` This is the default for retrieval. When data is retrieved, the persistent and non-persistent fields are merged together, with the persistent fields taking precedence over the non-persistent fields. You can view this as the persistent fields ”covering’ the non-persistent fields. You can also retrieve just the ”persistent’ fields, or just the ”non persistent’ fields.

◆ ◆ ◆ CHAPTER 31

Federated Search samples in Search SDK

This chapter describes the federated search sample programs and files included in the Search SDK.

Federated Search samples in Search SDK

The following federated search sample programs and files are included in the Search SDK:

`sampledbs.search`	The sample Search file for setting up federated search to these four types of database servers: ■ Google (using Google API) ■ Relational Database (using JDBC connection) ■ Directory Server (using JNDI connection) ■ Remote RDM Server (using Search SDK)
Database connector Java programs	Each is the Java class to connect to the target database server using designated APIs and submit search query: ■ `GoogleDb.java` ■ `JNDIDb.java` ■ `JDBCDb.java` ■ `RemoteRDMDb.java`
Search result set Java programs	Each is the Java class to manage search results and convert the results to RDM search results (Search format): ■ `GoogleResultSet.java` ■ `JNDIResultSet.java` ■ `JDBCResultSet.java` ■ `RemoteRDMResultSet.java`

`build.xml`	Compile the above Java files and build a `federatedsearch.jar`.

◆ ◆ ◆ CHAPTER 32

Developing a New Database or Search Engine Connector

This chapter includes instructions for developing a new database/search engine connector.

Developing a New Search Engine Connector

Federated search feature in Portal 7 supports the 4 types of federated search but not limited to these 4 types. Users can develop a new federated search connector following the steps described in this section.

▼ To Develop a New Federated Search Connector

1 **Implement the Database Connector class.**

Develop the Java interface class to connect to the new database server/search engine. This Java class must implement the RDMDb interface and implement the necessary methods (refer to any of the sample connector classes).

2 **Implement the Search Result class.**

Develop the Result Set Java class to manage returned results and convert the results to Search format for display. This Java class must extend RDMResultSet and implement result handling methods accordingly (refer to any of the sample result set classes).

3 **Modify `build.xml` file and compile your code.**

Once you have generated the JAR file, add your JAR file to the web container class path. If the new database server/search engine requires certain vendor-specific API library (for example, `googleapi.jar` for Google), add the library to the class path as well. The sample programs developed for this feature are already included in `searchserver.jar` file.

4 **Generate database configuration data.**

Add the configuration data for the new database server/search engine to the `sampledbs.search` file. Follow the attribute naming convention and Search syntax when editing the file. The configuration data is used by the Database Connector class to connect to the target server and submit a query.

5 **Develop front-end Search interface.**

Modify the Search channel JSP accordingly to add your new database name. The name must be the same name used in `sampledbs.search` file.

◆ ◆ ◆ CHAPTER 33

Localization: Templates and JSPs

This chapter contains the following sections:

This chapter addresses the Desktop templates and JSPs in the Portal Server software that can be translated to support localization.

Desktop Templates

This section contains:

Template Files

The `/var/opt/SUNWportal/portals/`*portal-ID*`/desktop/`*desktoptype* directory contains templates that are used to generate the Desktop pages. For each directory contained in this directory, a new directory for the locale must be created. The directory name must contain the existing directory with an underscore and the name of the locale; for example, `/var/opt/SUNWportal/portals/`*portal-ID*`/desktop/default` is translated to `/var/opt/SUNWportal/portals/`*portal-ID*`/desktop/default_`*locale*.

All of the file names can remain the same. Only files that require translation must be created in the locale-specific directory. If a file is not found in the locale-specific directory, it is automatically taken from the directory that does not have the locale specifier.

Template files are mostly markup text with embedded tags that specify substitutions to be performed at runtime. The tags have the form `[...]`. The tags should not be modified during the translation process.

Each template file is encoded using Java Unicode encoding where non-ASCII characters are represented with `\\uXXXX` notation. Here the `\\uXXXX` is the hexidecimal representation of the Unicode value for the character. This type of file can be created from a native file using the Java `native2ascii` program available in the JDK 1.3.1 package that is shipped with the Portal Server software.

JavaServer Pages

JSP files end with the `.jsp` suffix. The encoding for each top-level JSP file is specified using a header at the top of the file. The header appears as the following:

```
<%@ page contentType="text/html; charset=encoding" %>
```

where *encoding* is the desired encoding such as UTF-8 or EUC-JP. Note that this is only for top-level JSP files. JSPs that are statically included by other JSPs must use the encoding of the including JSP. That is, JSPs that are statically included by other JSPs using

```
<%@ include file="relativeURL" %>
```

must use the encoding of the including JSP. JSPs that are dynamically included using:

```
<jsp:include page="{ relativeURL | <%= expression %>}" flush="true" />
```

must have their own page directive that contains the encoding.

Image Files

If image files referenced by the existing templates are inappropriate for a locale, the references can be changed to refer to new files. The new file must be placed in the document root of the web container at *PortalServer-base*`/web-src` directory and then deploy it with the *PortalServer-base*`/bin/psadmin deploy` sub command. Or, you can deploy a new `WAR` file into the location of your choice.

File Lookup Mechanism

Desktop content can be customized based on the following factors:

- Desktop type
- Locale
- Client type

The DesktopServlet performs a search for a localized file before selecting a default. In order to perform the search, the DesktopServlet uses the user's locale attribute in the user's directory entry (or `preferredLocale` attribute in the Sun Java System Access Manager) and searches for the file. The user's directory entry stores the user's preferred locale in the `preferredLocale` attribute. The value of this attribute must be in the standard International Organization of Standards (ISO) locale format. For example, for the United Kingdom, this code is `en_UK`.

Locales are searched from more to less specific. For example, for a locale setting of `en_US_WE`, the search order would be: `en_US_WE`, `en_US`, `en`. If the locale-specific property is not found, then the non-locale-specific version is returned (if it exists).

The `/var/opt/SUNWportal/portals/`*portal-ID*`/desktop` directory stores the directories that present the desktop content. In the administration console, this is referred to as desktop type. The portal customizer can include different sets of files to alter the appearance of the desktop for different groups including localizations. The portal customizer must create a directory that contains the localized content at the same location (as the default content) and append the ISO locale code to the directory name.

◆ ◆ ◆ CHAPTER 34

Localization: Properties

This chapter contains the following sections:

Resource Bundles

This section contains:

Introduction to Resource Bundles

Resource bundles contain locale-specific objects. When your program needs a locale-specific resource, a String for example, your program can load it from the resource bundle that is appropriate for the current user's locale. See the documentation on Resource Bundles at `http://java.sun.com/products/jdk/1.2/docs/api/java/util/ResourceBundle.html` for more information on:

- The structure of the Resource Bundle
- The search look-up mechanism of Resource Bundles

The Desktop and the Desktop Providers use Java Resource Bundles for the localized on screen text.

Resource bundle for providers store content that is not included in display templates (such as error messages and dynamic content). The Portal Server software automatically loads the correct resource bundle for the user's locale (see Resource Bundle Access for more details).

File Naming Convention

The file name for the resource bundle is usually the provider name, that is, the name defined in the display profile when the provider class is declared. For example:

```
<Provider name="Bookmark" class="com.iplanet.portalserver.
providers.bookmark.BookmarkProvider">
```

Here, for the BookmarkProvider, the resource bundle file name is `Bookmark.properties`. The provider name will be retrieved via the `ProviderContext.getProviderName()` method.

The provider can also use a different name (different from the default name) for the resource bundle. The name must be passed as a key when calling `ProviderAdapter.getResourceBundle(String key)` method.

File Installation Location

The resource bundle files must be installed in the `/var/opt/SUNWportal/portals/`*portal-ID*`/desktop/classes` directory. For each `X.properties` file, create a `X_locale.properties` file that contains the translation. The translated file must be installed into the same directory as the English version.

File Entries Format

Each line of the properties file uses the format `key=value`. Lines can be continued using a backslash (\\) at the end of the line. The `.properties` files are encoded using Java Unicode encoding where non-ASCII characters are represented using `\\uXXXX` notation. Here, the `\\uXXXX` is the hexidecimal representation of the Unicode value for the character. This type of file can be created from a native file using the Java `native2ascii` program available in the JDK 1.3.1 package that is installed in the `JDK_DIR` directory specified in `/etc/opt/SUNWps/PSConfig.properties` file.

The messages represented by the value part of the line are formatted using the conventions of the Java `java.text.Message` format class. See the Javadocs for this class for more details on how to express parameters within messages. Parameters are enclosed in curly braces (`{}`).

Resource Bundle Access

The `ProviderAdapter.getResourceBundle()` method loads the correct resource bundle for the user's locale and if the specified locale is not found, it loads the default resource bundle. That is, it will look for the *providerName_locale*`.properties` file; if it is unable to find the specified properties file, it will look for *providerName*`.properties` file and load it.

Display Profile Properties

The display profile can be modified by using XML files that are input to the `dpadmin` command. The encoding for display profile XML file is specified using the following XML header at the top of the file:

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

where encoding is whatever encoding is suitable for the content of the file. Typically, the encoding is UTF-8; but it can also be other native encodings such as SJIS or ISO-8859-1. The DesktopServlet always converts characters internally to Unicode and stores them in the directory using UTF-8.

Every XML fragment must include a XML header and a doctype declaration as shown below:

```
<?xml version="1.0" encoding="encoding" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">
```

The display profile contains text strings for items such as channel titles and descriptions. These strings can be localized using the ConditionalProperties locale element.

For example, the user information channel is defined by the following element in the display profile:

```
<Channel name="UserInfo" provider="UserInfoProvider">
    <Properties>
        <String name="refreshTime" value="60" advanced="true"/>
        <String name="title" value="User Information"/>
        <String name="description" value="View/Edit User Information"/>
    </Properties>
</Channel>
```

▼ To Localize the User Information Channel Display Profile

1 **To localize the user information channel (title and description), the conditional properties tag for the locale must be added. For example, the following modifications must be included (modifications are shown in bold):**

```
<Channel name="UserInfo" provider="UserInfoProvider">
    <Properties>
        <String name="refreshTime" value="60" advanced="true"/>
        <String name="title" value="User Information"/>
        <String name="description" value="View/Edit
        User Information"/>

            <ConditionalProperties condition="locale" value="fr" >

            <String name="title" value="User Information in French"/>

            <String name="description" value="View/Edit User Information in French"/>

            </ConditionalProperties>
    </Properties>
</Channel>
```

2 **To add this, you need the following sample display profile fragment (stored in file** `dp-locale.xml`**).**

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">
    <ConditionalProperties condition="locale" value="fr" >
        <String name="title" value="User Information in French"/>
        <String name="description" value="View/Edit User Information in French"/>
    </ConditionalProperties>
```

3 **Upload the display profile fragment using the** `psadmin` **with the** `add-display-profile` **sub command.**

See the Command Line Reference for more information on this sub command.

◆ ◆ ◆ CHAPTER 35

35 Localization Support in PAPI

This chapter lists methods in the PAPI that support localization.

Localization Support in ProviderContext

The ProviderContext contains localization-aware methods such as:

- `getCollectionProperty(java.lang.String Channel,java.lang.String name, boolean localized)` and `getCollectionProperty(java.lang.String channel, java.lang.String name, java.util.Map def, boolean localized)` which gets the localized version of a collection property
- `getStringProperty(java.lang.String channel, java.lang.String name, boolean localized)` and `getStringProperty(java.lang.String Channel,java.lang.String name, java.lang.String def, boolean localized)` which gets the localized version of a String property
- `getCollectionProperty(String channel, String name, List pflist)`, `getCollectionProperty(String channel, String name, Map def, List pflist)`, `getStringProperty(String channel, String name, List pflist)`, `getStringProperty(String channel, String name, String def, List pflist)` to support conditional properties

See the Javadocs for more information on these methods.

Localization Support in ProfileProviderAdapter

The ProfileProviderAdapter contains localization-aware methods such as:

- `getMapProperty(java.lang.String key, boolean localized)` and `getMapProperty(java.lang.String key, java.lang.String def, boolean localized)` which gets the localized version of a map property for the channel.
- `getStringProperty(java.lang.String key, boolean localized)` and `getStringProperty(java.lang.String key, java.lang.String def, boolean localized)` which gets a localized string property for the channel.

  If localized is `true`, then this method will attempt to find a localized version of the map/string named by the key. The locale for the user who this object is executing is read from the ProviderContext object associated with this provider object.
- `getStringProperty(String channel, String name, List pflist)`, `getStringProperty(String channel, String name, String def, List pflist)`, `getMapProperty(String key, List pflist)`, `getMapProperty(String key, Map def, List pflist)` to support conditional properties.