



Sun Java™ System

Portal Server 6 Developer's Guide

2004Q2

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

Part No: 817-5319-10

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

THIS PRODUCT CONTAINS CONFIDENTIAL INFORMATION AND TRADE SECRETS OF SUN MICROSYSTEMS, INC. USE, DISCLOSURE OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SUN MICROSYSTEMS, INC.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and in other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Java, Solaris, JDK, Java Naming and Directory Interface, JavaMail, JavaHelp, J2SE, iPlanet, the Duke logo, the Java Coffee Cup logo, the Solaris logo, the SunTone Certified logo and the Sun ONE logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc.

Legato and the Legato logo are registered trademarks, and Legato NetWorker, are trademarks or registered trademarks of Legato Systems, Inc. The Netscape Communications Corp logo is a trademark or registered trademark of Netscape Communications Corporation.

The OPEN LOOK and Sun(TM) Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Products covered by and information contained in this service manual are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuelle relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plus des brevets américains listés à l'adresse <http://www.sun.com/patents> et un ou les brevets supplémentaires ou les applications de brevet en attente aux Etats - Unis et dans les autres pays.

CE PRODUIT CONTIENT DES INFORMATIONS CONFIDENTIELLES ET DES SECRETS COMMERCIAUX DE SUN MICROSYSTEMS, INC. SON UTILISATION, SA DIVULGATION ET SA REPRODUCTION SONT INTERDITES SANS L'AUTORISATION EXPRESSE, ECRITE ET PREALABLE DE SUN MICROSYSTEMS, INC.

Cette distribution peut comprendre des composants développés par des tierces parties.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, Java, Solaris, JDK, Java Naming and Directory Interface, JavaMail, JavaHelp, J2SE, iPlanet, le logo Duke, le logo Java Coffee Cup, le logo Solaris, le logo SunTone Certified et le logo Sun[tm] ONE sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

Legato, le logo Legato, et Legato NetWorker sont des marques de fabrique ou des marques déposées de Legato Systems, Inc. Le logo Netscape Communications Corp est une marque de fabrique ou une marque déposée de Netscape Communications Corporation.

L'interface d'utilisation graphique OPEN LOOK et Sun(TM) a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui, en outre, se conforment aux licences écrites de Sun.

Les produits qui font l'objet de ce manuel d'entretien et les informations qu'il contient sont regis par la législation américaine en matière de contrôle des exportations et peuvent être soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes biologiques et chimiques ou du nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou reexportations vers des pays sous embargo des Etats-Unis, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régi par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.

Contents

About This Guide	9
Who Should Read This Book	9
What You Need to Know	9
How This Book is Organized	10
Document Conventions Used in This Guide	11
Monospaced Font	11
Bold Monospaced Font	12
Italicized Font	12
Command-Line Prompts	12
Where to Find Related Information	12
Where to Find This Guide Online	13
Chapter 1 Introduction to the APIs	15
Overview	15
What Is a Portal?	16
Desktop and Desktop Hierarchy	16
Display Profile	17
Search Engine	18
Software Services	18
Application Development	19
Software Architecture	19
Overview of the APIs	21
Desktop APIs	21
Provider API	22
Building Blocks	23
Desktop Servlet	23
How Concepts in the Provider API Map to the Identity Server Software	24

Search APIs	24
Authentication APIs	26
Portal Environment	26
Web Container	27
File System	27

Part I Overview **29**

Chapter 2 Overview of the Provider API (PAPI)	31
Introduction	31
The Provider API	33
Provider	33
ProviderAdapter	33
ProfileProviderAdapter	34
ProviderContext	34
ProviderWidths	36
ProviderEditTypes	36
PropertiesFilter	37
Exceptions	37
The Provider Life Cycle	38
Chapter 3 Overview of the Desktop Servlet	45
Introduction	45
DesktopServlet Actions	46
Action content	47
Action edit	49
Action process	56
Action logout	60
Chapter 4 Overview of the Leaf Providers	61
Introduction	61
JSPProvider	62
URLScraperProvider	63
XMLProvider	64
Chapter 5 Overview of the Container Providers	65
The ContainerProvider Architecture	65
Overview of the ContainerProviders	67
ContainerProvider	67
ContainerProviderContext	67

ContainerProviderAdapter	68
JSPContainerProviderAdapter	68
JSPSingleContainerProvider	69
JSPTableContainerProvider	69
TabContainer	70
JSPTabContainerProvider	71
Chapter 6 Overview of Implementing a Custom Provider	73
Introduction	73
Defining Specific Requirements and Functionality	74
Application Specific Requirements	74
Presentation Method	75
Provider-specific Properties	77
Developing the Provider Class File	79
Provider Class File Location	80
Provider Class File	80
Provider Class Loader	80
Creating XML Fragments for Display Profile	82
Installing the Provider	84
Installing Manually	84
Transporting Using the par Utility	85
Part II Extending the Providers	87
Chapter 7 Extending the Base Providers	89
Implementing the Provider Interface	89
Extending the ProviderAdapter Class	93
Extending the ProfileProviderAdapter Class	96
Example 1	97
Example 2	103
Extending the PropertiesFilter Class	113
Chapter 8 Extending the Leaf Providers	117
Extending the JSPPProvider	117
Example 1	117
Example 2	121
Extending the URLScrapperProvider	124
Extending the XMLProvider	128

Chapter 9 Extending the Container Providers	133
Extending the JSPSingleContainerProvider	133
Extending the JSPTableContainerProvider	137
Extending the JSPTabContainerProvider	161
Chapter 10 Creating a Custom ContainerProvider	169
Part III Portlets	179
Chapter 11 Overview of the Portlets	181
What is a Portlet?	181
Overview of Developing and Deploying Portlets	181
Chapter 12 Extending the GenericPortlet Abstract Class	183
Developing the Class File	183
Compiling the Portlet	185
Creating a Portlet Web Application	185
Deploying the Application	187
Creating Channels from the Deployed Portlets	187
Chapter 13 Converting Providers to Portlets	189
Developing the Class File	189
Mapping ProviderAdapter to GenericPortlet	190
Sample Code Fragments for Provider to Portlet Conversion	191
Compiling the Portlet	193
Creating a Portlet Web Application	193
Deploying the Application	195
Creating Channels from the Deployed Portlets	195
Part IV WSRP	197
Chapter 14 Validating Registration Data	199
Overview	199
Registering with the Producer	200
Validating the Registration Data	200
The RegistrationValidator Interface	200
Default Implementation	201

Chapter 15 Defining Custom Registration Validators	203
Implementing the RegistrationValidator Interface	203
Installing the Class File	205
Customizing the Return Codes	205
Configuring the Producer's Registration Validator	206
Part V Search Engine Robot	207
Chapter 16 Search Engine Robot Overview	209
Introduction	209
How the Robot Works	210
Robot Configuration Files	211
The Filtering Process	211
Stages in the Filter Process	212
Chapter 17 Robot Completion Scripts	215
Introduction	215
Monitoring cmdHook Execution	216
Preparing Your Completion Script to Appear in the Administration Interface	216
Chapter 18 Overview of the Robot Application Functions	219
Introduction	219
Robot Plug-in API Overview	220
The Robot Application Function Header Files	220
Header File Hierarchy	221
Header File Contents	221
Chapter 19 Writing New Robot Application Functions	225
Introduction	225
RAF Prototype	225
Writing Functions for Specific Directives	226
Passing Parameters to Robot Application Functions	226
Working with Parameter Blocks	228
Getting Information on the Processed Resource	229
Returning a Response Status Code	230
Reporting Errors to the Robot Log File	231
RAF Definition Example	232
Compiling and Linking your Code	234
Loading Your Shared Object	235
Using your New Robot Application Functions	235

Part VI Using the SOIF API to Work with SOIF Objects	237
Chapter 20 Overview of the SOIF API	239
Introduction	239
What is SOIF?	240
Using the SOIF API	240
An Introductory Example	241
Getting Search Server Database Contents as a SOIFStream	243
Chapter 21 SOIF API	245
Functions and Objects	245
SOIF Structure	247
Attribute-Value Pair Routines	250
Multi-valued Attribute Routines	252
Stream Routines for Parsing and Printing SOIFs	255
Filtering SOIF Objects	258
Memory Buffer Management	259
Part VII Using the RDM API to Access the Search Engine and Database in C	261
Chapter 22 What is an RDM?	263
Introduction	263
RDM Format Syntax	263
RDM Header	264
RDM Body	264
Chapter 23 About the RDM API	267
Introduction	267
rdm.h File	268
API Reference	269
Finding the RDM Version	269
Creating and Freeing RDM Structures	269
RDMHeader	269
RDMQuery	272
Other RDM Structures	273
Chapter 24 Example of Submitting a Query	275
Introduction	275
RDMHeader and RDMQuery Object Parameters	275
Running the Example	277

Part VIII Using Java To Access the Search Engine Database	281
Chapter 25 Overview of the Search Engine Java SDK	283
Introduction	283
The Search Engine Java SDK	283
Running the Sample Applications	284
Chapter 26 Using Java To Access the Search Server Database	287
Creating a Search Object	287
Executing A Query and Getting the Results	289
Working Through An Example Search Application	290
Import the Necessary Classes	290
Define the SearchDemo Class	290
Define the SimpleSearch Class	292
Execute the Search Query	294
Display the Results	295
Chapter 27 Using Java To Add Entries to the Search Engine Database	297
rdmgr	297
SOIF Object	298
Constructing and Submitting a Request	298
Constructing a Request	298
Submitting a Request	300
Part IX Localizing Sun Java System Portal Server	303
Chapter 28 Templates and JSPs	305
Desktop Templates	305
Template Files	305
JSPs	306
Images	307
File Lookup Mechanism	307
Chapter 29 Properties	309
Resource Bundles	309
File Naming Convention	310
File Installation Location	310
File Entries Format	310
Resource Bundle Access	311
Display Profile	311

Chapter 30 Localization Support in PAPI	313
Glossary	315
Index	1

About This Guide

This guide explains how to use the Sun Java™ System Portal Server software APIs. Portal Server software provides a platform to create portals for your organization's integrated data, knowledge management, and applications. Portal Server software platform offers a complete infrastructure solution for building and deploying all types of portals, including business-to-business, business-to-employee, and business-to-consumer.

This preface includes the following sections:

- [Who Should Read This Book](#)
- [What You Need to Know](#)
- [How This Book is Organized](#)
- [Document Conventions Used in This Guide](#)
- [Where to Find Related Information](#)
- [Where to Find This Guide Online](#)

Who Should Read This Book

You should read this book if you are responsible for customizing and extending the Portal Server software APIs.

What You Need to Know

This book assumes that you are familiar with the following products:

- Sun Java™ System Directory Server

- Sun Java™ System Identity Server
- Sun Java™ System Web Server
- Sun Java™ System Application Server

This book assumes you already know the basics of:

- The Solaris Operating Environment and UNIX command-line utilities and administrative tasks
- One or more of the following programming languages: Java, JavaScript™, C, XML, and HTML

How This Book is Organized

This book contains the following chapters:

About This Guide (this chapter)

Chapter 1, “Introduction to the APIs” This chapter provides an introduction to the Portal Server APIs.

Part I, “Overview” This part includes chapters that provide an overview of the various public extensible set of classes in the Portal Server software. The chapters provide:

- An introduction to the base PAPI interfaces, classes, exceptions, and an overview of the DesktopServlet, and introduction to the building-block leaf and container providers.
- A description of the development process for a Portal Server software provider. It touches on the public APIs available for integrating an application with the Portal Server software Desktop via the provider.

Part II, “Extending the Providers” This part includes chapters that provide instructions for:

- Implementing a custom provider by extending the PAPI base classes.
- Customizing the methods in the leaf provider and container provider classes.
- Developing a custom ContainerProvider and with sample instructions for developing a custom RouterContainerProvider.

Part III, “Portlets” This part includes chapters that provide:

- An overview of the portlets
- Instructions for developing portlets by extending the GenericPortlet class.
- Information for converting providers to portlets

Part V, “Search Engine Robot” This part includes chapters that describe the:

- Search Engine Robot
- Application Programming Interface (API) used to create plug-in robot application functions (RAFs).

Part VI, “Using the SOIF API to Work with SOIF Objects” This part includes chapters that describe the use of the search engine SOIF API to work with SOIF objects in C.

Part VII, “Using the RDM API to Access the Search Engine and Database in C” This part includes chapters that describe the use of the Search Engine RDM API to access the Search Engine and its database.

Part VIII, “Using Java To Access the Search Engine Database” This part includes chapters that describe how to:

- Submit queries
- Add entries to the Search Engine database by using the Java programming language.

Part IX, “Localizing Sun Java System Portal Server” The part includes chapters that provide instructions for localizing the customizable parts of the Portal Server software.

Document Conventions Used in This Guide

Monospaced Font

Monospaced font is used for any text that appears on the computer screen or text that you should type. It is also used for file names, distinguished names, functions, and examples.

Bold Monospaced Font

Bold monospaced font is used to represent text within a code example that you should type. For example, you might see something like this:

```
portal-server-install-root/SUNWps/bin/dpadmin add -p parent -u amadmin_dn -w  
password -d node dp-locale.xml
```

Italicized Font

Italicized font is used to represent text that you enter using information that is unique to your installation (for example, variables). It is used for server paths and names and account IDs.

Command-Line Prompts

Command-line prompts (for example, % for a C-Shell, or \$ for a Korn, or Bourne shell) are not displayed in the examples. Depending on which operating system environment you are using, you will see a variety of different command-line prompts. However, you should enter the command as it appears in the document unless specifically noted otherwise.

Where to Find Related Information

In addition to this guide, the Portal Server software comes with supplementary information for administrators as well as documentation for developers. Use the following URL to see all the Portal Server software documentation:

<http://docs.sun.com/prod/slportalsrv>

Use this book in conjunction with the Javadocs at <http://hostname:port/URI/javadocs>. Listed below are the additional documents that are available:

- *Portal Server Deployment Planning Guide*
- *Portal Server Migration Guide*
- *Portal Server Administration Guide*
- *Portal Server, Secure Remote Access Administration Guide*
- *Portal Server Desktop Customization Guide*

- *Sun Java System Portal Server 6 2004Q2 Release Notes*

Where to Find This Guide Online

You can find the *Sun Java System Portal Server 6 Developer's Guide* online in PDF and HTML formats. This book can be found at the following URL:

<http://docs.sun.com/prod/portalsrv>

Introduction to the APIs

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 software architecture, APIs, and programming concepts that developers can use.

This chapter contains the following sections:

- [Overview](#)
- [Overview of the APIs](#)
- [Portal Environment](#)

See the *Portal Server Administration Guide* for information on administering the Portal Server software. See the *Portal Server Desktop Customization Guide* for information on customizing the Desktop templates and JavaServer Pages™ (JSP™.)

NOTE Detailed information on the Portal Server software APIs is available in the Javadocs. The URL to access the Javadocs is:

`http://hostname:port/portal/javadocs`

Overview

This section provides a brief description of the Portal Server software. See *Portal Server Administration Guide* 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 Portal Server software 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.

Desktop and Desktop Hierarchy

The Desktop uses 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.

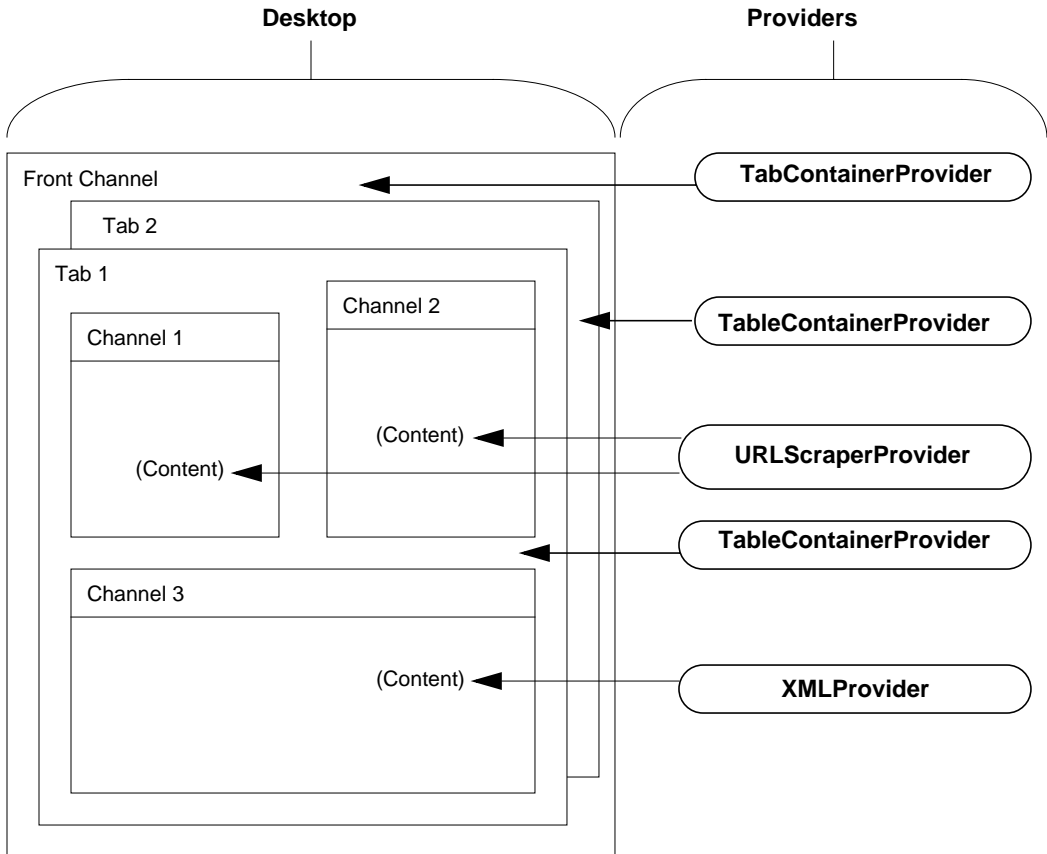
In the Desktop, leaf channels are the basic unit of content, displaying a specific type of information. To the end user, a channel is a distinct unit of content in the Desktop, usually (but not always) set off with a border and header row of icons that enables users to configure the channel to their 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.

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.

See [Figure 1-1 on page 16](#) for 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, URLScrapperProvider, is serving more than one channel. Providers can have a one-to-many relationship with channels.

Figure 1-1 Desktop Hierarchy and Building Block Providers



Display Profile

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

- 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 Identity Server software services layer. That is, the display profile documents are an LDAP attribute residing in an instance of the Sun Java System Directory Server software.

See the *Portal Server Administration Guide* and *Portal Server Desktop Customization Guide* for a complete discussion of the display profile and its administrative tasks.

Search Engine

The Portal Server software provides a search engine service that includes 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 SOIF (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.

Software Services

Portal applications and resources use Identity Server software to provide services such as authentication, single-sign-on, logging, profile, and session management. See the *Identity Server Administration Guide* for more information.

Application Development

As a developer, you can provide access to portal resources through the Portal Server software (and the Identity Server software) 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 software, you 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 Identity Server software - There are five APIs packaged with the Identity Server software: 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 Identity Server software APIs in detail, but instead provides a general overview. See the *Identity Server Programmer's Guide* for more information on programming with those APIs.

Software Architecture

[Figure 1-2 on page 20](#) shows the Portal Server software architecture. In this figure, the Portal Server software components consist of:

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

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

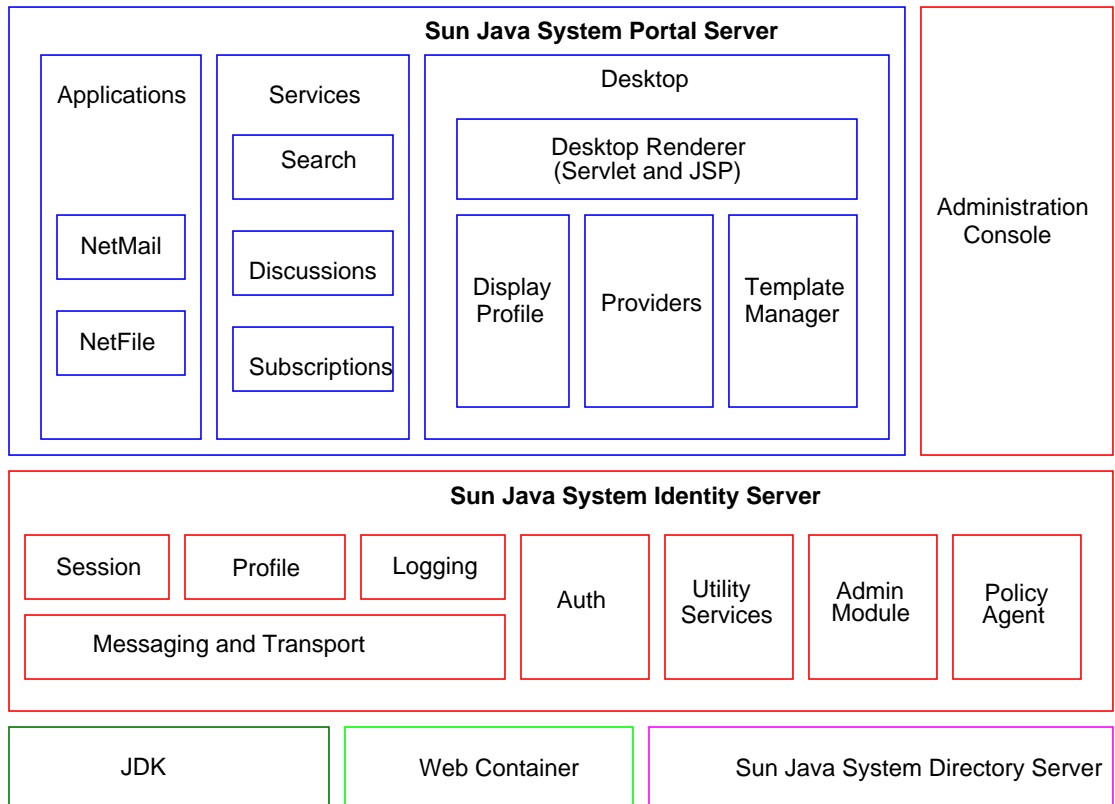
The Sun Java System Web Server software, or the Sun Java System Application Server software, or the BEA WebLogic Server, or the IBM Websphere Application Server provides the web container. The application server provides the Portal Server with a robust, highly scalable web container for the Portal Server 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 software provides the user profile data repository. The Identity Server software provides support for core services such as profile, session, authentication, and logging. It also provides single-sign-on services, policy management, debug utility, the administration console, and client support interfaces for the Portal Server. Use the Identity Server software administration console for service, profile, and user administration. See the *Identity Server Administration Guide* and *Portal Server Administration Guide* for more information.

The Portal Server software 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 Content 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 Identity Server service for storing content provider and channel data. The Desktop also includes an administration console module for editing the display profile and other Desktop service data.

Figure 1-2 Portal Server Software Architecture



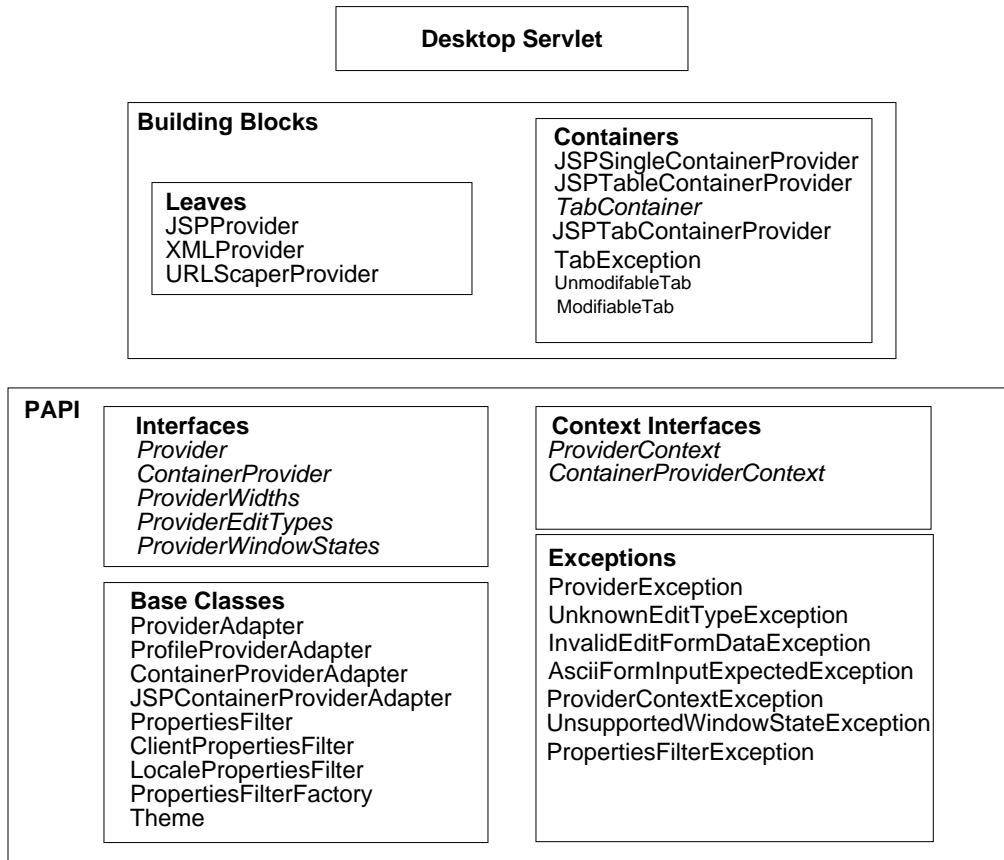
Overview of the APIs

This section describes the Portal Server software 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 [Figure 1-3](#).

Figure 1-3 Desktop APIs



Provider API

At the bottom of [Figure 1-3](#) 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. See chapters 2 through 4 for more information.

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.

Building Blocks

At the next level of [Figure 1-3 on page 21](#) 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 software 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.

See the *Portal Server Desktop Customization Guide* for more information on the out-of-the box content providers.

Desktop Servlet

At the top of [Figure 1-3 on page 21](#) 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 Identity Server Software

The Provider API furnishes architectural separation from the Identity Server software; but the PAPI is implemented in terms of specific Identity Server software APIs within the Portal Server software 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 Provider and ContainerProviderContext interfaces. The implementation of these interfaces connects to Identity Server's software services in an implementation independent manner. The actual implementation of the 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 software 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 SOIF objects, such as RDs (RDM and SOIF 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 Identity Server software administration console.

The robot has many customizable 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 (SOIF) 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 SOIF output stream, and read a SOIF input stream.

Normally, the Search database can be accessed by using the SOIF 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. SOIF is the format used to represent RDs.

Authentication APIs

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

In general, most development work for a portal developer in the Identity Server software area will be to customize the authentication interfaces. [Table 1-1](#) 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	<i>Identity Server Administration Guide</i>
Enable or disable authentication modules	<i>Identity Server Administration Guide</i>
Add a custom authentication module	<i>Identity Server Programmer's Guide</i> . By default, the Identity Server software supplies authentication modules for the following types of logins: Anonymous, Certificate, LDAP, Membership, RADIUS, SafeWord, SecureID, and UNIX.

Portal Environment

This section contains the following:

- [Web Container](#)
- [File System](#)

Web Container

The Portal Server software providers run within the Java™ Virtual Machine provided by the web container, which may vary between different web containers. The Portal Server software environment in the Sun Java System Web Server software or Sun Java System Application Server software web container is Java™ Development Kit (JDK™) 1.3.1_04.

File System

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

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

The Portal Server software installs all of its web applications code into a single web application. By default, this web application is installed on the `/portal` URI in the web container. The Portal Server web applications are built dynamically by the `deploy` command after the individual files that comprise the web application are installed onto the system.

Overview

Chapter 2, “Overview of the Provider API (PAPI)”

Chapter 3, “Overview of the Desktop Servlet”

Chapter 4, “Overview of the Leaf Providers”

Chapter 5, “Overview of the Container Providers”

Chapter 6, “Overview of Implementing a Custom Provider”

Overview of the Provider API (PAPI)

This chapter provides an overview of the Sun Java System Portal Server software Provider Application Programming Interface (PAPI) and includes instructions for developing a custom provider.

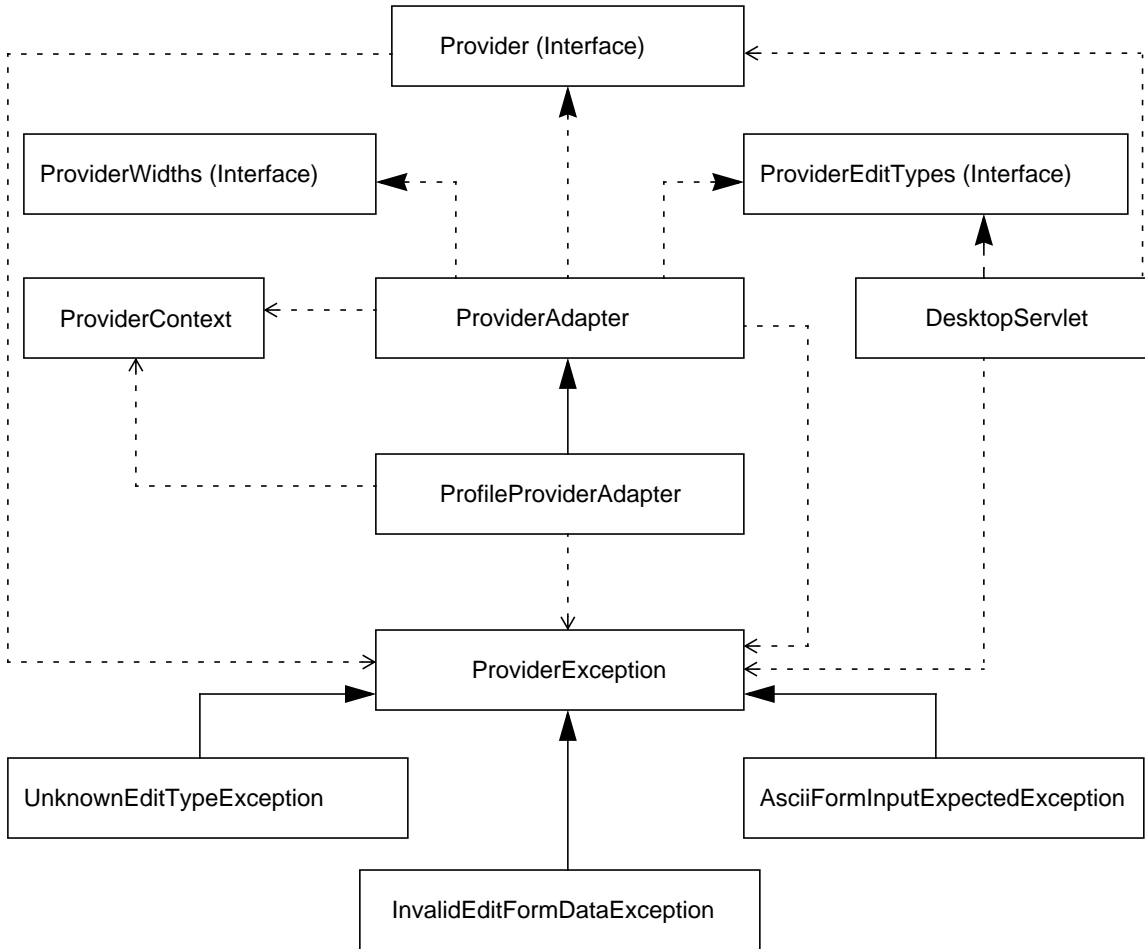
This chapter contains the following sections:

- [Introduction](#)
- [The Provider API](#)
- [The Provider Life Cycle](#)

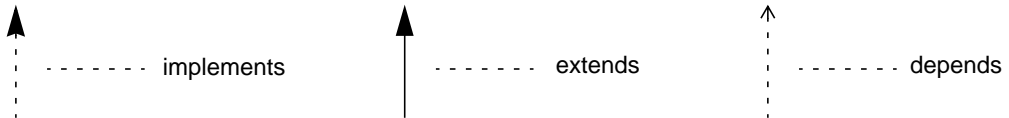
Introduction

[Figure 2-1](#) 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 <http://hostname:port/portal/javadocs>.

Figure 2-1 The Provider Application Programming Interface (PAPI)



Legend:



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

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.

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

The `ProviderAdapter` class implements aspects of the `Provider` using the `ProviderContext`. 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

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

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 following are some of the methods in the ProviderContext:

- **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()`, `isDebugEnabled()`, `isDebugEnabled()`, `isDebugEnabled()`, `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 config 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

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

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

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.

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 `/var/opt/SUNWam/debug/desktop.debug` file. Note that this file is created only if there is an error.

ProviderException

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

AsciiFormInputExpectedException

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

InvalidEditFormDataException

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

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 on page 39](#), [Figure 2-3 on page 42](#), and [Figure 2-4 on page 42](#) 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 on page 39](#), 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 on page 42](#) and [Figure 2-4 on page 42](#) 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 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 on page 42](#), 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 on page 39](#) and [Figure 2-4 on page 42](#) 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 “.”

Figure 2-2 The Provider Life Cycle - Initial Request for Authenticated User

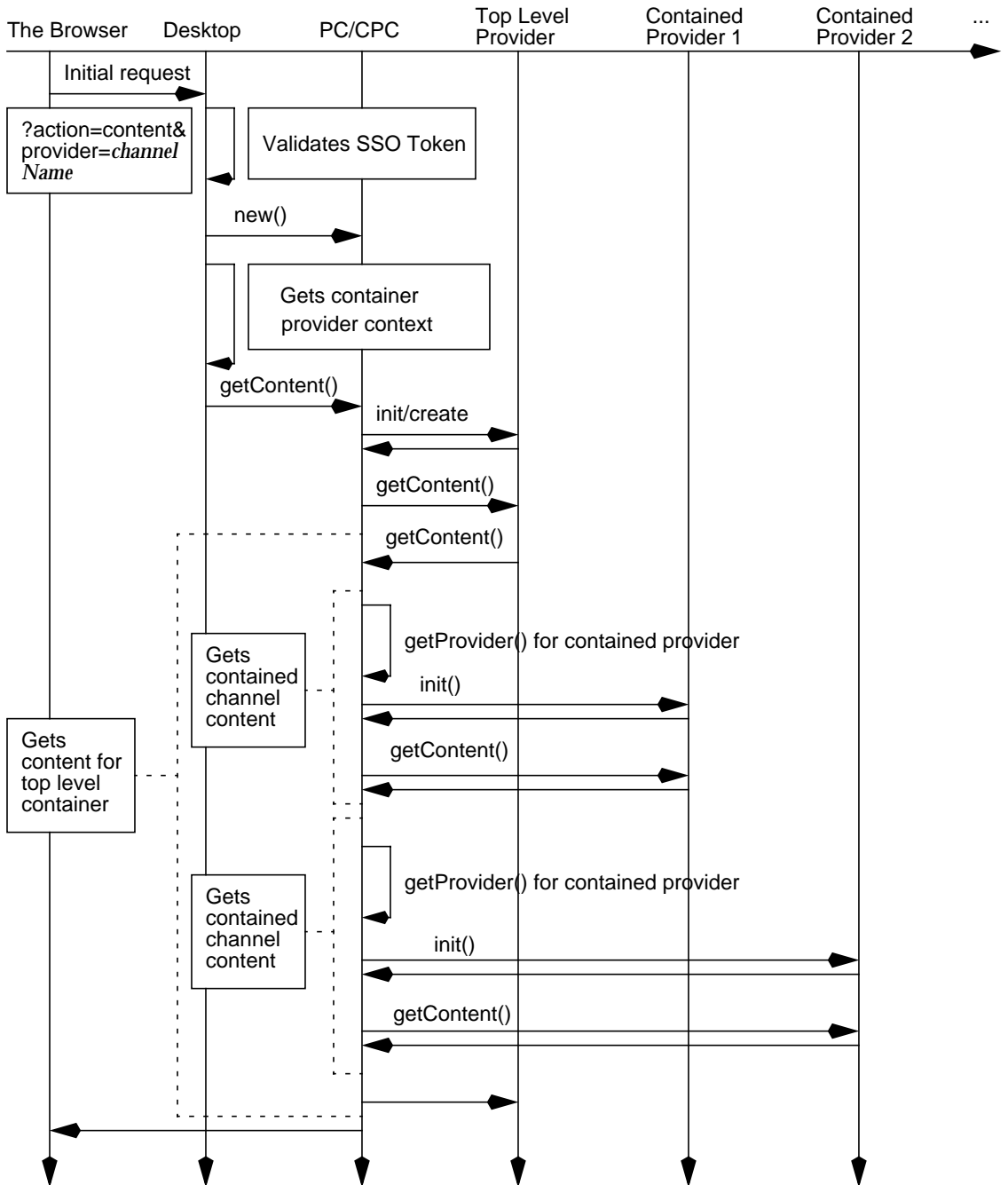


Figure 2-3 The Provider Lifecycle - Desktop Reload Request for Authenticated User

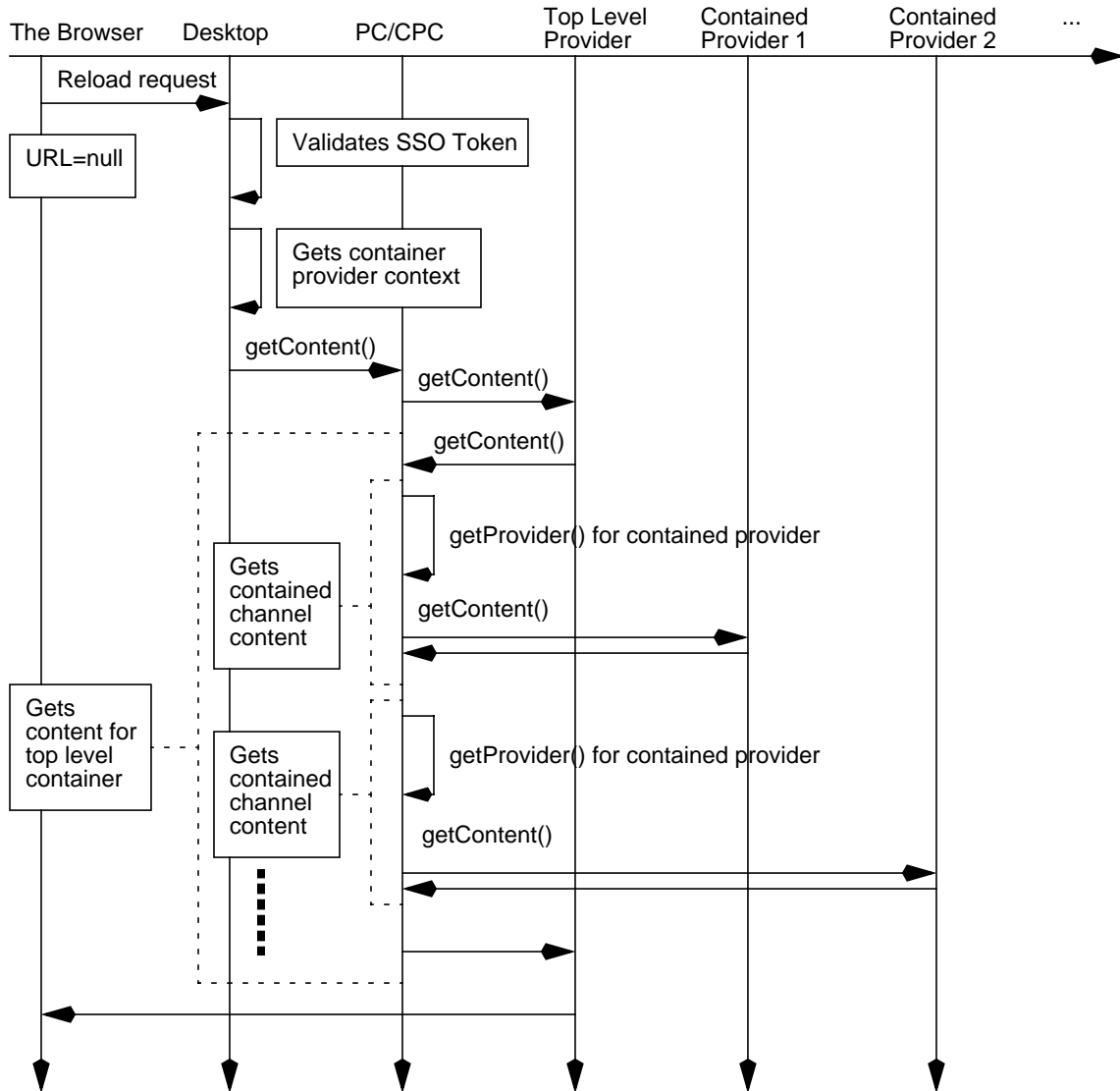
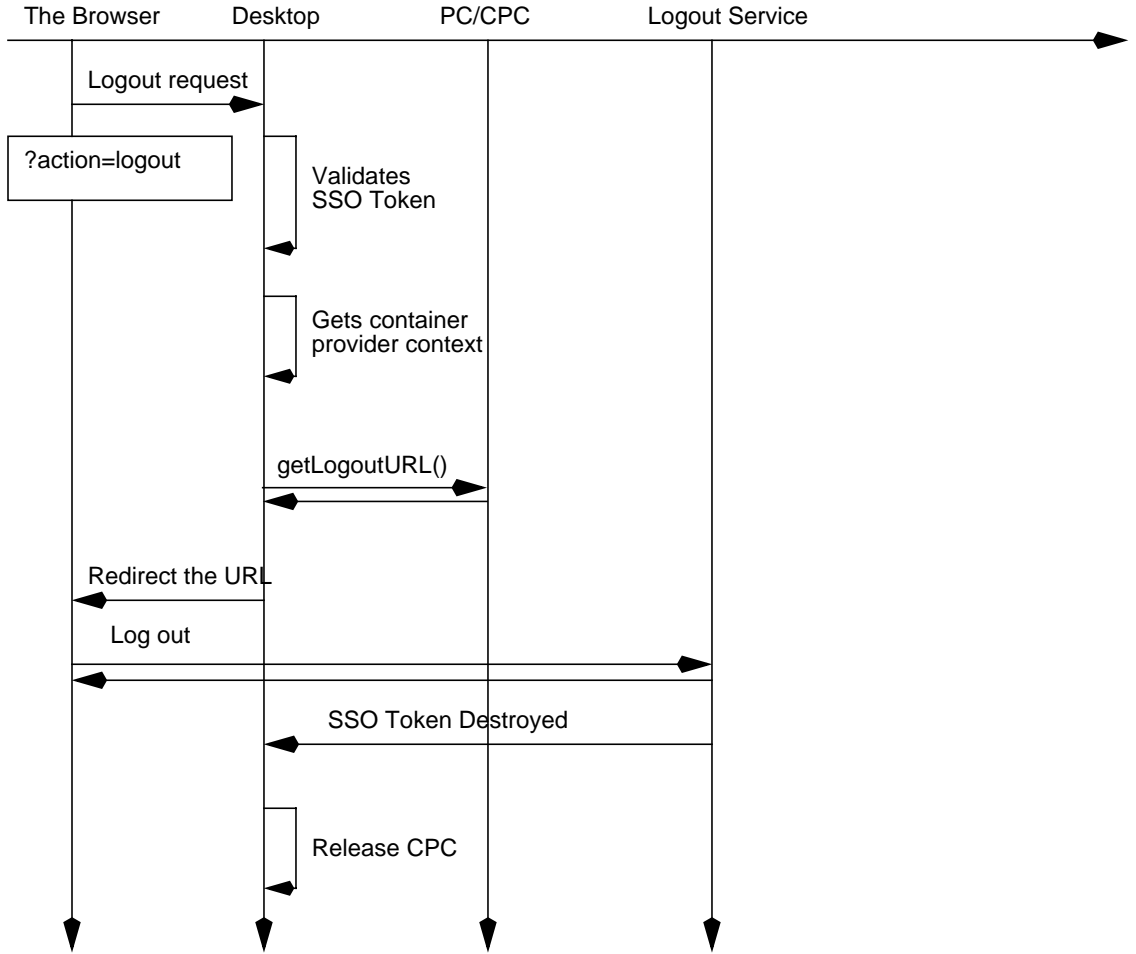


Figure 2-4 The Provider Lifecycle - Logout Request



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](#)
- [DesktopServlet Actions](#)

Introduction

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 Identity Server 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.”](#) The following sections only describe how the DesktopServlet handles the 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, pass in 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 Identity Server 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.

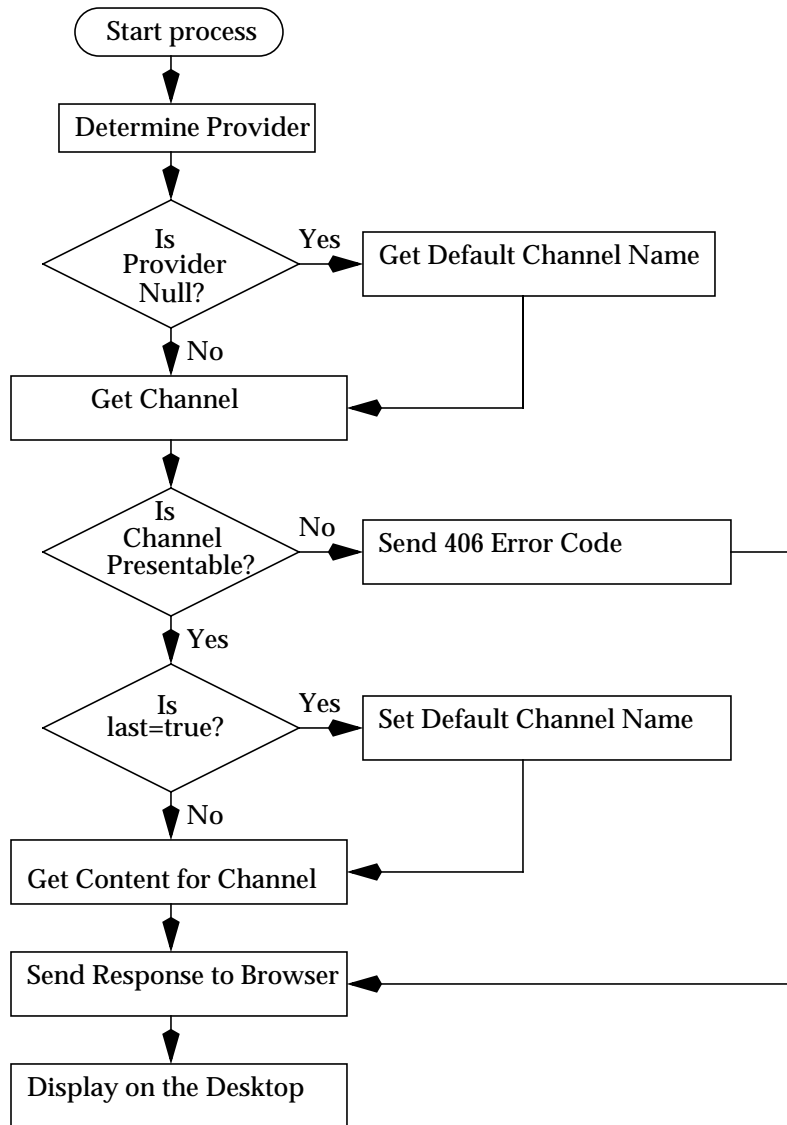
2. Once the provider is determined, the provider's `isPresentable()` method is invoked to determine if the provider can be presented to the requesting client.

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.

DesktopServletURL?action=edit&provider=ChannelName

The flowchart in [Figure 3-2 on page 50](#) 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. Default or 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.

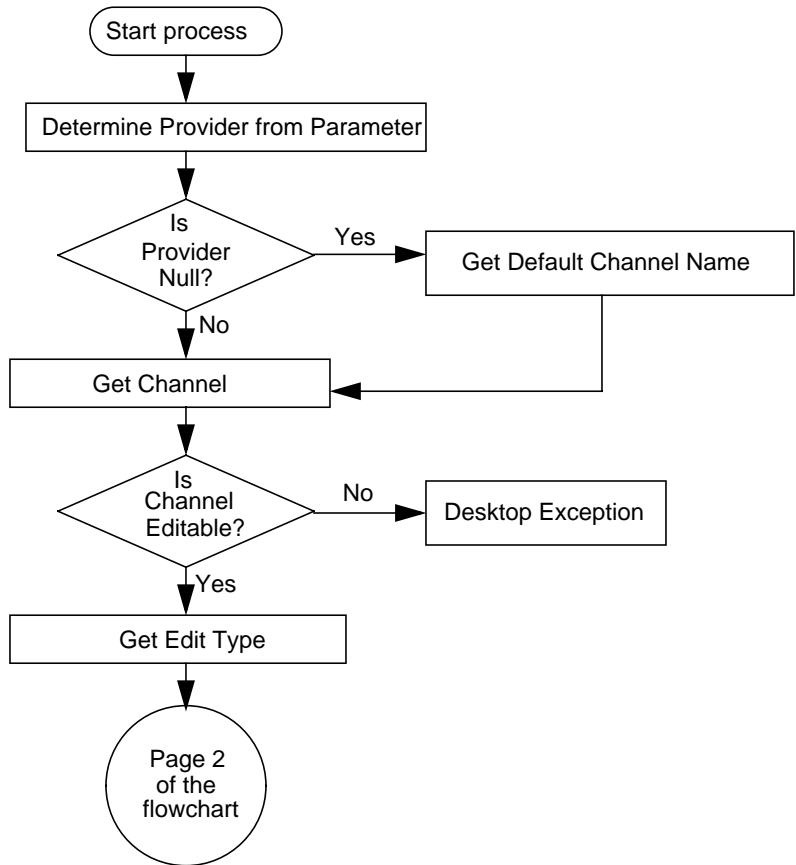
3. If the channel's edit type is:
 - `EDIT_COMPLETE`, the provider's `getEdit()` method is invoked and the Edit page for the channel is returned on the Desktop.
 - `EDIT_SUBSET`, the edit container's provider name will be detected (via the default Desktop Edit Container attribute), and then the edit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

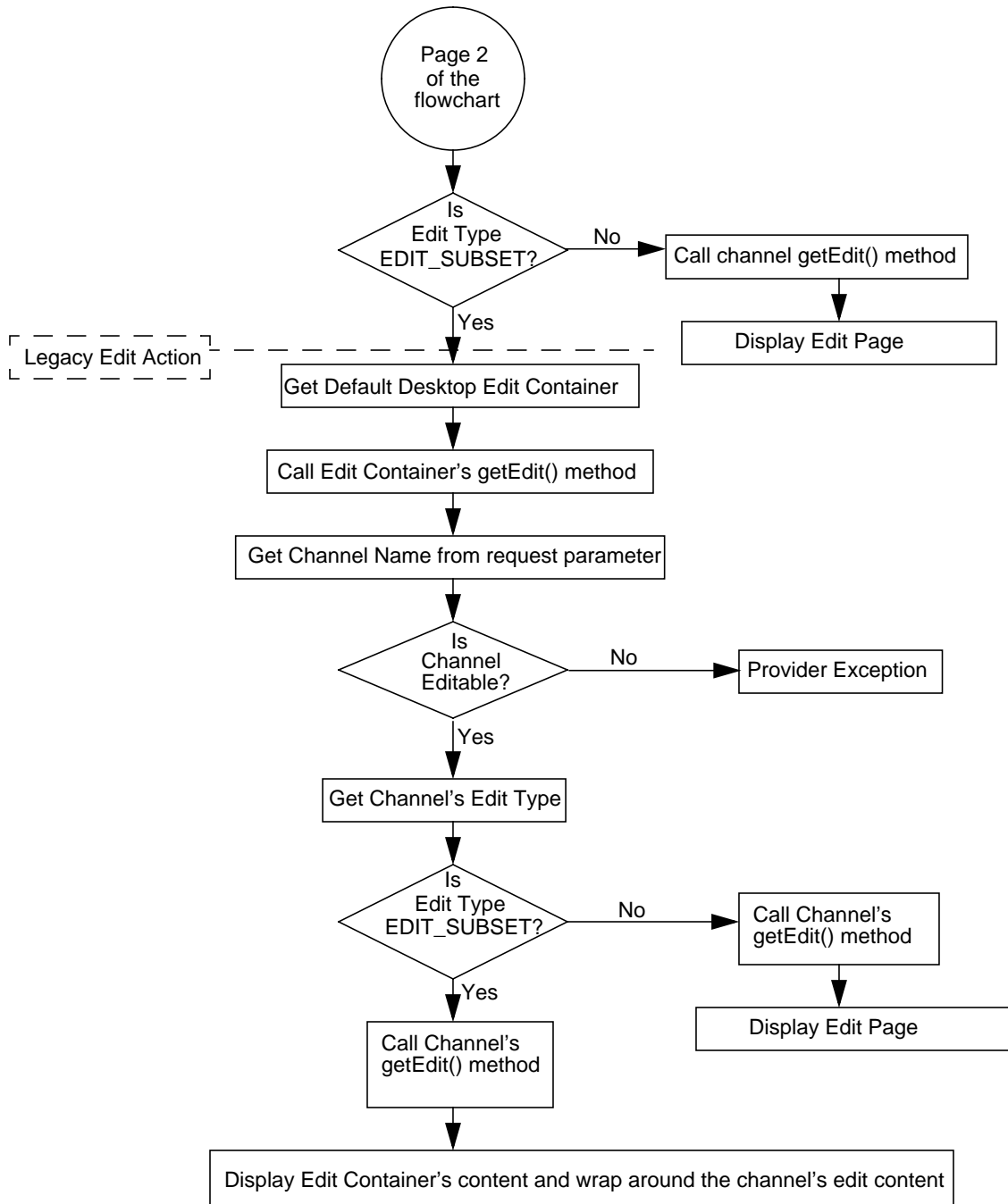
- I. The edit container's provides a common look and feel of the edit page for all the channels that it contains. After it generates the markup for the common look and feel, it detects the channel name from the request parameter and determines whether the target channel is editable.

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.

- II. 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 DesktopServlet Legacy edit Action





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

The flowchart in [Figure 3-3 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. Default or 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:

- o `EDIT_COMPLETE`:

- I. The edit container's `getEdit` method is invoked and the leaf channel's name is determined from the request parameter.
- II. 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.

- III. If the leaf channel's edit type is:

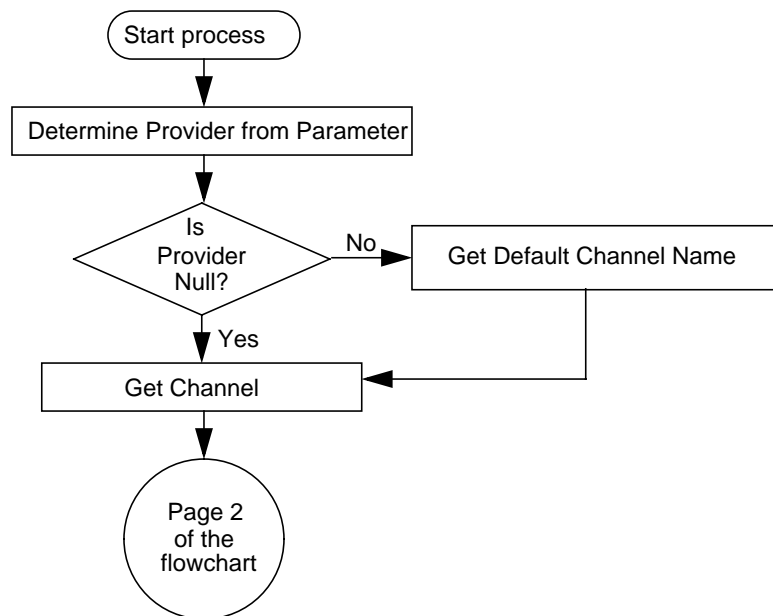
- `EDIT_COMPLETE`, the container calls the leaf channel's `getEdit()` method and displays the edit page on the Desktop.

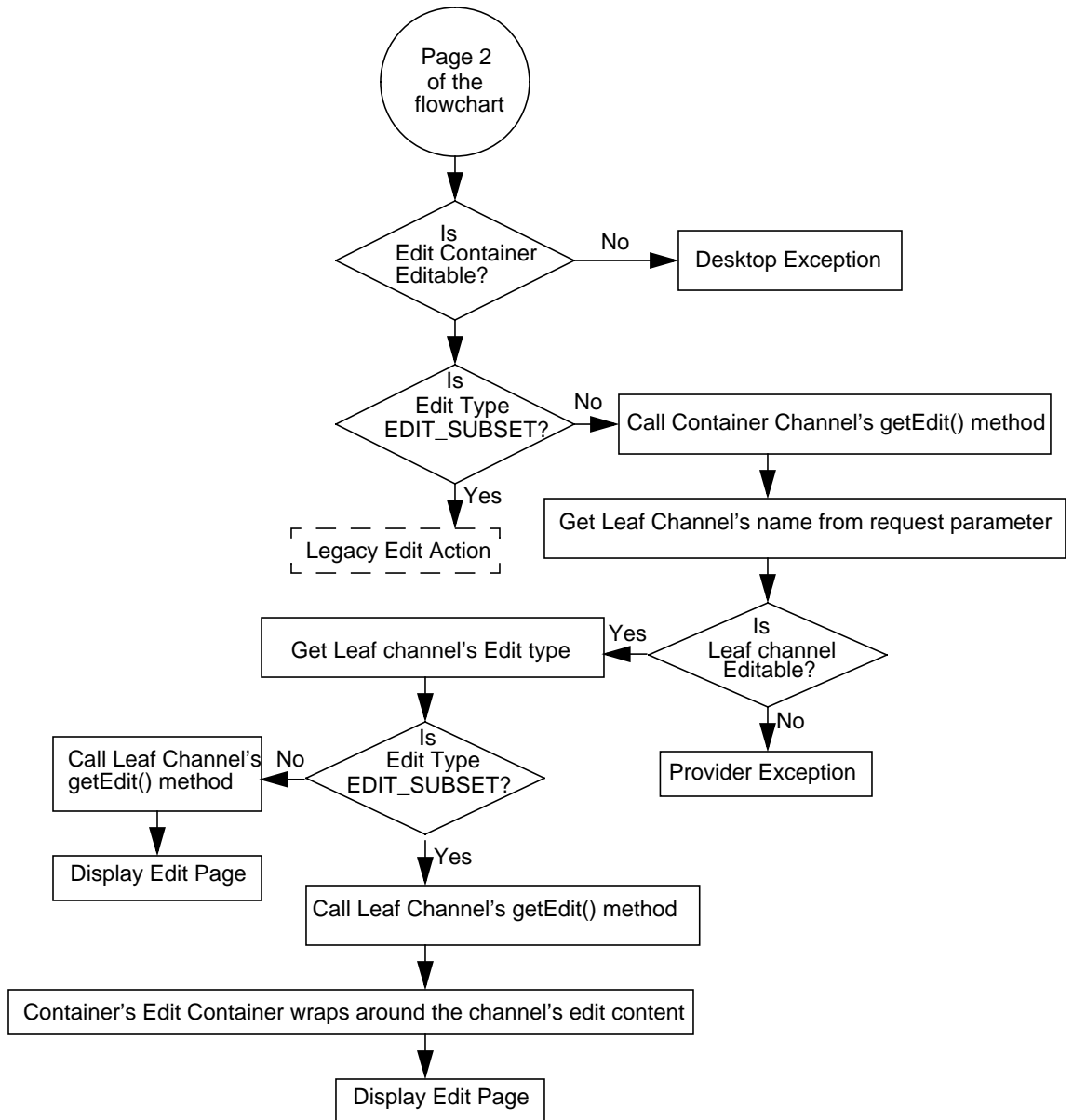
- `EDIT_SUBSET`, the container calls the leaf channel's `getEdit()` method and the container's Edit Container wraps the leaf channel's content before displaying the Edit page on the Desktop.

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 Identity Server 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 [page 49](#) 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

DesktopServletURL?action=process&provider=channelName

When the DesktopServlet receives a request where the action is `process` (see [Figure 3-4](#)), the DesktopServlet:

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

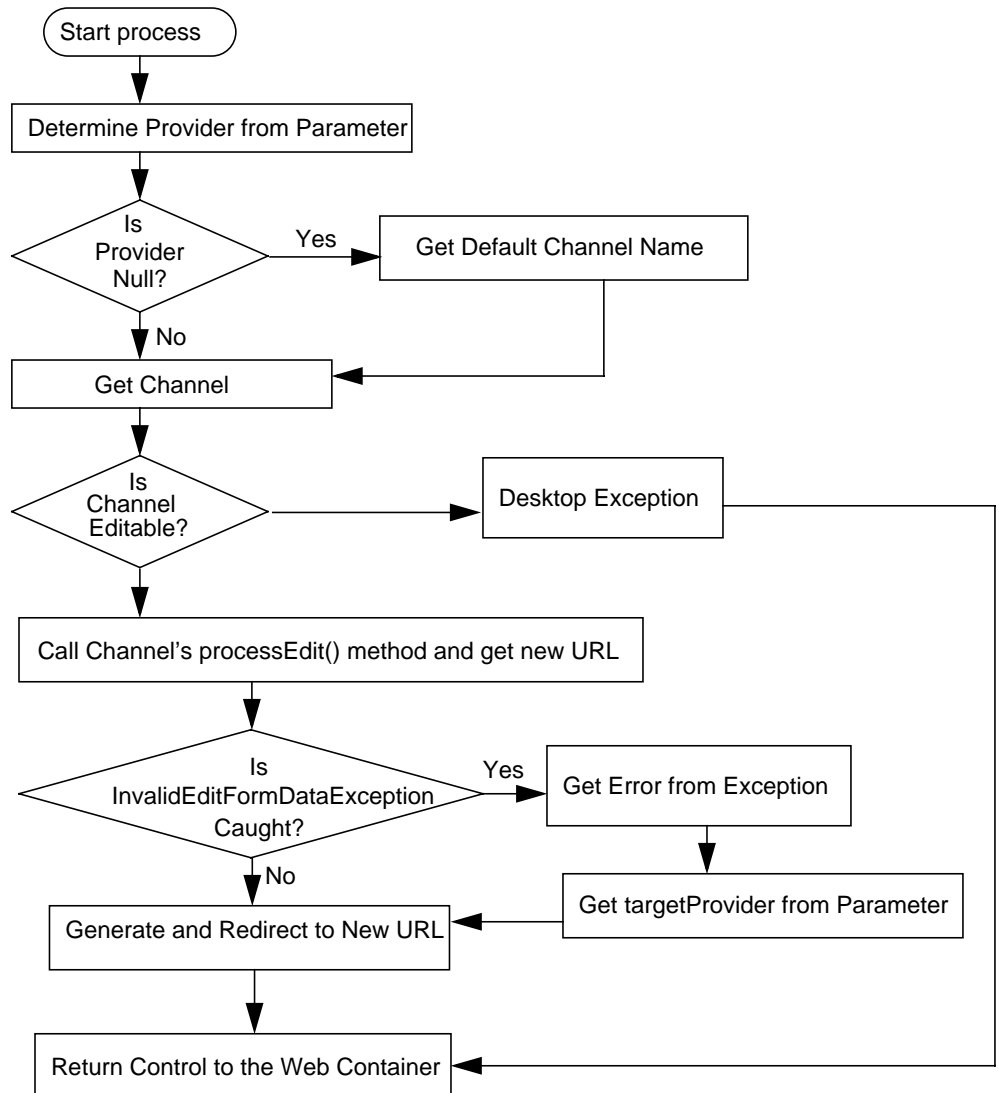
The `processEdit()` method is called to process the edit page generated from the `getEdit()` method. The request passed in contains the parameters.

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



`DesktopServletURL?action=process&provider=editContainer&targetprovider=channel Name`

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 [Figure 3-5 on page 58](#)), 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.

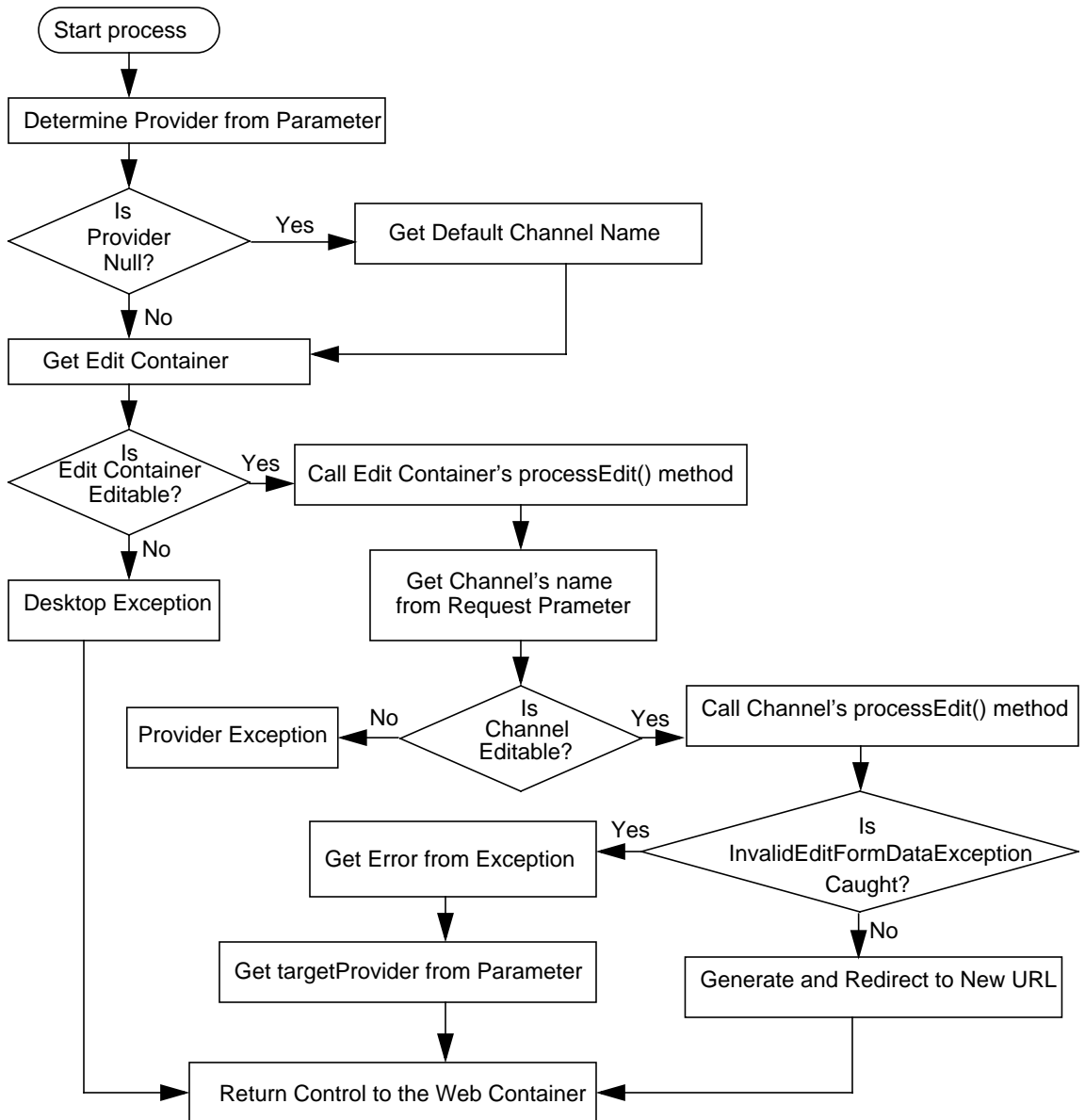
3. Re-directs to the URL returned from the provider's `processEdit()` method.

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 Identity Server 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.

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.

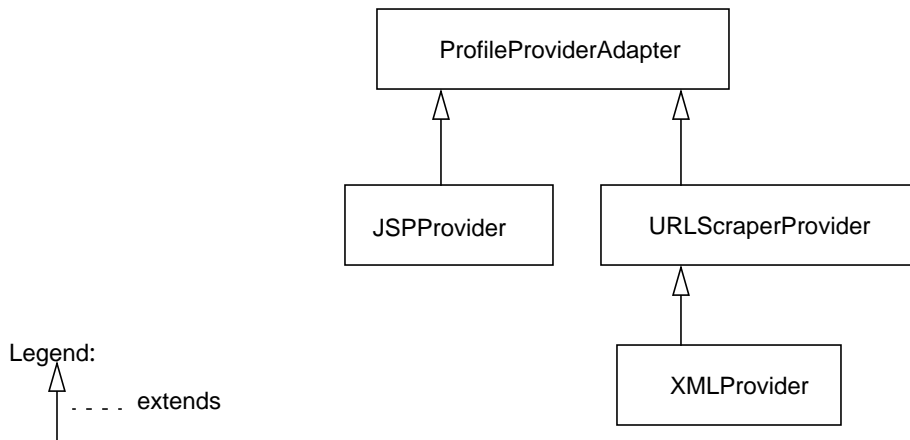
This chapter contains the following:

- [Introduction](#)
- [JSPPProvider](#)
- [URLScrapperProvider](#)
- [XMLProvider](#)

Introduction

[Figure 4-1 on page 61](#) shows the relationship between the PAPI base class and the leaf providers discussed in this chapter. The JSPPProvider and URLScrapperProvider extend the ProfileProviderAdapter to include support for retrieving content from JavaServer Pages and URLs respectively. The XMLProvider extends the URLScrapperProvider'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 URLScrapperProvider's built-in functionality to fetch the XML contents from HTTP, HTTPS, and file URLs.

Figure 4-1 The Building Block Providers



JSPPProvider

A JSPPProvider is a content provider that can use JavaServer Pages to create the content for a channel on the Desktop.

The following is a description of the step-by-step process of the JSPPProvider invoking a JSP.

1. The `JSPPProvider.getContent()` method is called and the JSPPProvider 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`.

URLScrapperProvider

The `URLScrapperProvider` can retrieve and display content from a given URL.

The following is a step-by-step description of how the `URLScrapperProvider` 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.

3. Gets the `urlScrapperRulesetID` to be used by `Rewriter` as a string.
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.
6. Gets the content by:

- Retrieving 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).

Or, by:

- Taking 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 exist 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.

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](#)
- [Overview of the ContainerProviders](#)
- [JSPContainerProviderAdapter](#)

The ContainerProvider Architecture

[Figure 5-1 on page 66](#) shows the relationship between the various interfaces and classes discussed in this chapter. The JSPProvider (see [Chapter 4, “Using Leaf Providers”](#) for more information) and ContainerProviderAdapter classes extend ProfileProviderAdapter (see [Chapter 2, “Using Provider API \(PAPI\)”](#) 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.

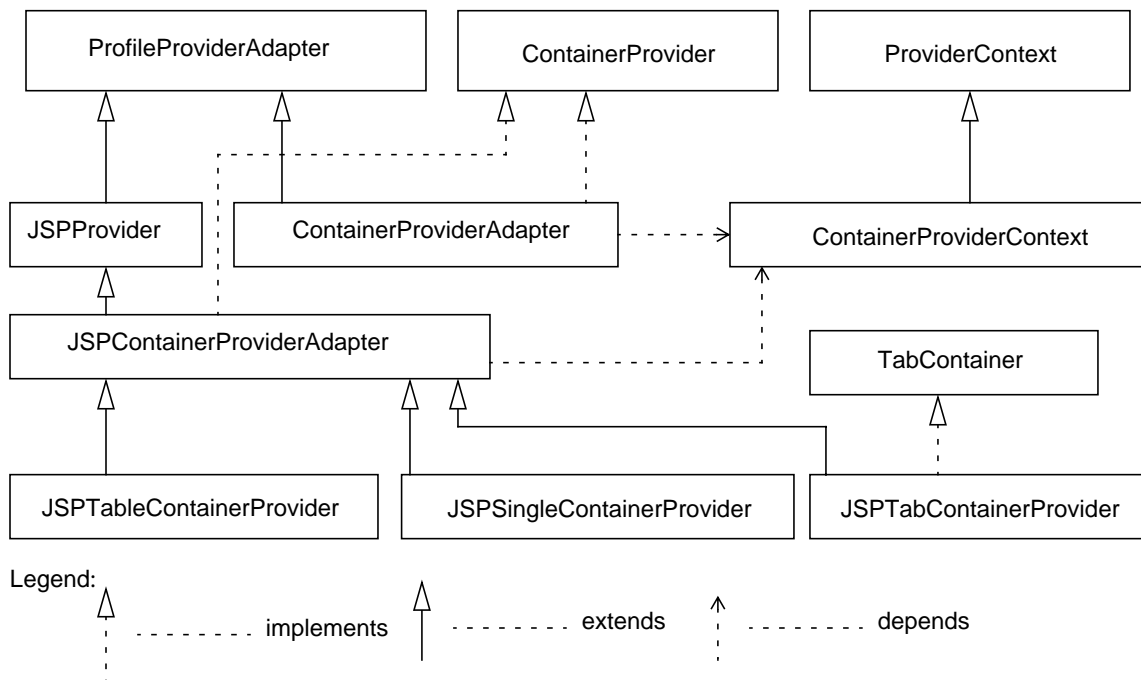
The ContainerProviderAdapter provides default implementations of the following methods in the ContainerProvider: `get/setSelectedChannels()`, `get/setAvailableChannels()`, `get/setWindowState()`, and `getSupportedWindowStates()`. For more information on all the methods in this class, see the Javadocs.

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, “Using Leaf Providers”](#) for more information) and provides implementations of methods in the ContainerProvider interface to facilitate the execution of JavaServer Pages™ (JSP™).

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

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

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, “Using Provider API \(PAPI\)”](#) 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

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

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](#), [JSPTableContainerProvider](#), and [JSPTabContainerProvider](#)) to the `JSPContainerProviderAdapter` API are included.

JSPSingleContainerProvider

Introduction

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 decorative 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

Introduction

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.

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

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

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

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.

Overview of Implementing a Custom Provider

This chapter contains the following sections:

- [Introduction](#)
- [Defining Specific Requirements and Functionality](#)
- [Developing the Provider Class File](#)
- [Creating XML Fragments for Display Profile](#)
- [Installing the Provider](#)

Introduction

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`).

To develop a provider, you must start by:

- [Defining Specific Requirements and Functionality](#) that includes defining:
 - Application Specific Requirements
 - Presentation Method
 - Provider-specific Properties

- [Developing the Provider Class File](#)
- [Creating XML Fragments for Display Profile](#)
- [Installing the Provider](#)

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:

*/etc/opt/SUNWps/desktop/**desktoptype**/**channeldirectory**/**templatefiles***

The JSPs and templates are stored under */etc/opt/SUNWps/desktop/default* and */etc/opt/SUNWps/desktop/sampleportal* directories. Storing the files in the */etc/opt/SUNWps/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 sub directory 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

templatroot

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
- *templatroot* is, by default, `/etc/opt/SUNWps/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/clientPath

desktoptype

default_locale/channelname

default_locale/provider

default_locale/clientPath

default_locale

default/channelname

default/provider

default/clientPath

default

templatroot

The root of the search directory (default value of `/etc/opt/SUNWps/desktop/`) can be changed by modifying the `templateBaseDir` property in the `desktopconfig.properties` file.

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”](#) and *Portal Server Desktop Customization Guide* 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 79 for more information.

Properties in Display Profile

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.

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

See *Portal Server Desktop Customization 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 *Portal Server Administration 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 *portal-server-install-root/SUNWps/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 80](#) 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 relative (non-fully qualified) 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.

Provider Class File Location

The provider classes must be deployed in the provider class base directory specified in the file `/etc/opt/SUNWps/desktop/desktopconfig.properties`. By default, the provider class base directory is `/etc/opt/SUNWps/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 API

(*identity-server-install-root*/SUNWam/lib/servlet.jar) and PAPI

(*portal-server-install-root*/SUNWps/sdk/desktop/desktopsdk.jar) are required in the CLASSPATH. That is, when compiling a **Provider.java* file, you must type:

```
javac -d /etc/opt/SUNWps/desktop/classes -classpath
portal-server-install-root/SUNWps/sdk/desktop/desktopsdk.jar : identity-server-install-root/SUNWam/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:

- `container1` of type `HeadlineContainerProvider` loaded by loader1
- `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 `/etc/opt/SUNWps/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 to 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 to 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 to the following structure:

```

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

```

To upload display profile changes, type:

- `dpadmin add -u dn_admin -w password -d distinguishedname provider.xml` if the display profile XML fragment was stored in a separate file
- `dpadmin modify -u dn_admin -w password -d distinguishedname provider.xml` if an existing display profile document was directly edited

where *provider.xml* contains the display profile XML fragment for the provider. See *Portal Server Administration Guide* for more information on the `dpadmin` command and on administering Portal Server software display profile.

Installing the Provider

This section provides information on:

- Installing a provider into a portal server instance:
 - Manually without using the `par` utility
 - Transporting using the `par` utility
- [Provider Class Loader](#)

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 `/etc/opt/SUNWps/desktop/classes`. Or copy the file to the location specified in the `/etc/opt/SUNWps/desktop/desktopconfig.properties` file.
2. Copy the resource bundle files, if any, to `/etc/opt/SUNWps/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 `/etc/opt/SUNWps/desktop/desktopype/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 par Utility

Utilize the `par` utility for deploying channels and providers. The `par` command enables you to transfer or move providers or channels from one the Portal Server software host to another. The `par` 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 and an 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 `jar` or `export` subcommand of the `par` utility.

To export an existing provider, use the `export` subcommand of the `par` 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.

```
par export -r dn_amadmin -p password -v provider.par dn exportfile
```

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

```
par import -r dn_admin -p password -v provider.par dn  
provider=providername,channel=channelname
```

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 *Portal Server Administration Guide* for more information on the `par` utility.

Extending the Providers

Chapter 7, “Extending the Base Providers”

Chapter 8, “Extending the Leaf Providers”

Chapter 9, “Extending the Container Providers”

Chapter 10, “Creating a Custom ContainerProvider”

Extending the Base Providers

This chapter includes instructions for creating a custom provider by:

- [Implementing the Provider Interface](#)
- [Extending the ProviderAdapter Class](#)
- [Extending the ProfileProviderAdapter Class](#)
- [Extending the PropertiesFilter Class](#)

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:

NOTE 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 in [Code Example 7-1](#).

Code Example 7-1 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;
    }
}
```

Code Example 7-1 HelloWorldProviderP.java File (Continued)

```
    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 `/etc/opt/SUNWps/desktop/classes`. To compile the `HelloWorldProviderP.java` file, type:

```
javac -d /etc/opt/SUNWps/desktop/classes -classpath
portal-server-install-root/SUNWps/sdk/desktop/desktopsdk.jar:identity-server-install-root/SUNWam/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` (see [Code Example 7-2](#)) and `HelloChannelP.xml` (see [Code Example 7-3](#)) files.

Code Example 7-2 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>
```

Code Example 7-3 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 `dpadmin` command.

For the sample `HelloWorldProvider`, upload the `HelloProviderP.xml` and `HelloChannelP.xml` XML fragments using the `dpadmin` command. That is, for example, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_admin -w password -d
distinguishedname HelloProviderP.xml
```

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_admin -w password -d
distinguishedname HelloChannelP.xml
```

5. Access the channel from a browser. To access, type the following URL in your browser:

```
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:

NOTE 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 in [Code Example 7-4 on page 94](#).

Code Example 7-4 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 `/etc/opt/SUNWps/desktop/classes`. To compile the `HelloWorldProviderPA.java` file, type:

```

javac -d /etc/opt/SUNWps/desktop/classes -classpath
portal-server-install-root/SUNWps/sdk/desktop/desktopsdk.jar:identity-server-install-root/SUNWam/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 in the `HelloProviderPA.xml` is shown in [Code Example 7-5](#).

Code Example 7-5 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>
  </Properties>
</Provider>

```


Code Example 7-5 HelloProviderPA.xml File (Continued)

```

<String name="helpURL" value="desktop/HelloWorld.htm" advanced="true"/>
<String name="fontFacel" 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 `dpadmin` command.

For the sample `HelloWorldProvider`, upload the `HelloProviderPA.xml` XML fragments using the `dpadmin` command. That is, for example, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_amadmin -w password -d
distinguishedname HelloProviderPA.xml
```

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 `JSPTTableContainer` from the administration console, follow instructions in section *Portal Server Administration Guide*. To add the channel in the `JSPTTableContainer` manually:

- a. Add the `HelloWorldProvider` channel XML fragment (in the `HelloChannelPA.xml` file) for the `JSPTTableContainer` as shown in [Code Example 7-6](#).

Code Example 7-6 HelloChannelPA.xml File

```

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

<Container name="JSPTTableContainer" provider="JSPTTableContainerProvider">
  <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>
</Container>

```

Code Example 7-6 HelloChannelPA.xml File (Continued)

```

</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 `dpadmin modify` subcommand 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.

```
portal-server-install-root/SUNWps/bin/dpadmin modify -u dn_amadmin -w password -d
distinguishedname HelloChannelPA.xml
```

6. Access the HelloWorld channel inside the JSPTTableContainer. To access, type the following URL in your browser:

```
http://hostname:port/portal/dt?provider=JSPTTableContainer/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:

Example 1

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 as shown in [Code Example 7-7](#).

Code Example 7-7 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 tags = new Hashtable();
        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 tags = new Hashtable();
        tags.put("message", getStringProperty("message"));
        tags.put("properties", "");
    }
}
```

Code Example 7-7 HelloWorldProviderPPA1.java File (Continued)

```

        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 `/etc/opt/SUNWps/desktop/classes`. To compile the `HelloWorldProviderPPA1.java` file, type:

```
javac -d /etc/opt/SUNWps/desktop/classes -classpath
portal-server-install-root/SUNWps/sdk/desktop/desktopsdk.jar :identity-server-install-root/SUNWam/lib/servlet.jar HelloWorldProviderPPA1.java
```

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

The sample `HelloWorldProvider` XML fragment for the provider in the `HelloProviderPPA1.xml` file and the XML fragment for the channel in the `HelloChannelPPA1.xml` file are shown in [Code Example 7-8](#) and [Code Example 7-9 on page 99](#) respectively.

Code Example 7-8 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>

```

Code Example 7-8 HelloProviderPPA1.xml File (Continued)

```

<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>

```

Code Example 7-9 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 `dpadmin` command.

For the sample `HelloWorldProvider`, upload the `HelloProviderPPA1.xml` file and `HelloChannelPPA1.xml` file XML fragments using the `dpadmin` command. That is, for example, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_amadmin -w password -d
distinguishedname HelloProviderPPA1.xml
```

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_amadmin -w password -d
distinguishedname HelloChannelPPA1.xml
```

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 Guide*. To add the channel in the JSPTableContainer manually:

- a. Add the HelloWorldProvider channel XML fragment (in the HelloContainerPPA1.xml file) for the JSPTableContainer as shown in [Code Example 7-10](#).

Code Example 7-10 HelloContainerPPA1.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="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 dpadmin modify subcommand 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.

```
portal-server-install-root/SUNWps/bin/dpadmin modify -m -u dn_amadmin
-w password -d distinguishedname HelloContainerPPA1.xml
```

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 in [Code Example 7-11](#).

Code Example 7-11 HelloWorldProviderPPA1 content.template File

```
<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 in [Code Example 7-12](#).

Code Example 7-12 HelloWorldProviderPPA1 edit.template File

```
<!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]
```

Code Example 7-12 HelloWorldProviderPPA1 edit.template File (Continued)

```

<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

`/etc/opt/SUNWps/desktop/desktopype`. For this example, create a HelloWorldPPA1 directory under `/etc/opt/SUNWps/desktop/desktopype` directory.

8. Copy all the template files over to the newly created directory (in Step 7). For example:

```

cp content.template /etc/opt/SUNWps/desktop/desktopype/HelloWorldPPA1
cp edit.template /etc/opt/SUNWps/desktop/desktopype/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!!
```

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

By default, the class directory to copy the resource bundle into is /etc/opt/SUNWps/desktop/classes.

11. Access the HelloWorld channel inside the JSPTableContainer. To access, log in to the Desktop and type the following URL in your browser:

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

Example 2

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 as shown in [Code Example 7-13](#).

Code Example 7-13 HelloWorldProviderPPA2.java File

```
package custom.helloworld;

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

import java.net.URL;
```

Code Example 7-13 HelloWorldProviderPPA2.java File (Continued)

```

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 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;
    }
}

```

Code Example 7-13 HelloWorldProviderPPA2.java File (Continued)

```

/*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 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 tags = new Hashtable();
    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));
}

```

Code Example 7-13 HelloWorldProviderPPA2.java File (Continued)

```

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.getStandardTags():
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 /etc/opt/SUNWps/desktop/classes -classpath
portal-server-install-root/SUNWps/sdk/desktop/desktopsdk.jar:identity-server-install-root/SUNWam/lib/servlet.jar HelloWorldProviderPPA2.java
```

3. Create the provider and channel definition for the display profile.

The `HelloWorldProvider` provider display profile XML fragment (in file `HelloProviderPPA2.xml`) is shown in [Code Example 7-14](#).

Code Example 7-14 HelloProviderPPA2.xml File

```
<?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 in [Code Example 7-15 on page 108](#).

Code Example 7-15 HelloChannelPPA2.xml File

```

<?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="fontFacel" 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>

```

4. Load the provider and channel display profile definitions using the `dpadmin` command.

To load the sample `HelloProviderPPA2` provider and channel definitions, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_amadmin -w password -d
distinguishedname HelloProviderPPA2.xml
```

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_amadmin -w password -d
distinguishedname HelloChannelPPA2.xml
```

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

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

6. Copy the resource bundle properties file, `HelloWorldProviderPPA2.properties`, into `/etc/opt/SUNWps/desktop/classes` directory.

7. Develop the template files for the provider.

The `HelloWorldProvider` requires the following template files:

- o `content.template` (see [Code Example 7-16 on page 109](#))
- o `edit.template` (see [Code Example 7-17 on page 109](#)) - This file generates the editable fields in the form

- o editTemplate.template (see [Code Example 7-18 on page 110](#)) - This file generates a complete form

Code Example 7-16 HelloWorldProviderPPA2 content.template File

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

Code Example 7-17 HelloWorldProviderPPA2 edit.template File

```
<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%">&nbsp;</td>
    <td width="45%">&nbsp;</td>
  </tr>

</table>
```

Code Example 7-18 HelloWorldProviderPPA2 editTemplate.template File

```

<!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:fontFacel]">
    <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:fontFacel]">
            <B>
            Edit [tag:title]</B></FONT>
            <BR>
            [tag:contentOptions]
          </CENTER>
        </TD>
      </TR>
    </TABLE>

    <BR>

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

```


Code Example 7-18 HelloWorldProviderPPA2 editTemplate.template File (Continued)

```

        </font>
    </BODY>
</HTML>

```

8. Create the channel directory under the template root directory.

By default, the template root directory is `/etc/opt/SUNWps/desktop/desktoptype`. For this example, create a `HelloWorldPPA2` directory under `/etc/opt/SUNWps/desktop/desktoptype` directory.

9. Copy the templates for the provider into the channel directory under the templates root directory. For example:

```

cp content.template /etc/opt/SUNWps/desktop/desktoptype/HelloWorldPPA2
cp edit.template /etc/opt/SUNWps/desktop/desktoptype/HelloWorldPPA2
cp editTemplate.template
/etc/opt/SUNWps/desktop/desktoptype/HelloWorldPPA2

```

10. Access the channel from a browser. To access, type the following URL in your browser:

```

http://hostname:port/portal/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:

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

For example, to add the HelloWorldProvider to the TemplateTableContainer, type the following:

- o To add to the Available list, type the following:

```

portal-server-install-root/SUNWps/bin/SUNWps/bin/dpadmin modify -m -p
"TemplateTableContainer" -u dn_admin -w password -d distinguishedname << EOF
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">
<Available>
<Reference value="HelloWorldPPA2"/>
</Available>
EOF

```

- o To add to the Selected list, type the following:

```
portal-server-install-root/SUNWps/bin/SUNWps/bin/dpadmin modify -m -p
"TemplateTableContainer" -u dn_amadmin -w password -d distinguishedname << EOF
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE DisplayProfile SYSTEM "jar://resources/psdp.dtd">
<Selected>
<Reference value="HelloWorldPPA2"/>
</Selected>
EOF
```

2. Access the HelloWorld Channel inside the TemplateTableContainer. To access, log in to the Desktop and type the following URL in your browser:

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

Utilize the `par` utility for transporting channels and providers. For example, the following command can be used to export the HelloWorld channel and provider into a `par` file which can be imported to another Portal Server system.

```
par export -r dn_amadmin -p password -v helloworld.par distinguishednode
helloworld.txt
```

Here, `helloworld.txt` contains the following:

```
from: channel+provider HelloWorldPPA2
file: providerClassBaseDir HelloWorldProviderPPA2.properties provider
class: custom.helloworld.HelloWorldProviderPPA2
directory: templateBaseDir . HelloWorldPPA2
desc: HelloWorld Provider
```

Using the following command, the contents of the `helloworld.par` can be imported to as HelloWorld channel in a different Portal Server software installation.

```
par import -r dn_amadmin -p password -v helloworld.par distinguishednode
provider=HelloWorldProvider,channel=HelloWorld,avail=TemplateTableContaine
r,selected
```

The above command, apart from installing the channel and provider, will also include the HelloWorld channel as one of the selected and available channels in the TemplateTableContainer. For more information on the `par` utility and its subcommands, see the *Portal Server Administration Guide*.

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.

This section provides the instructions for creating 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 in [Code Example 7-19](#).

Code Example 7-19 DateLaterThanPropertiesFilter.java File

```
import com.acme.filters.DateLaterThanPropertiesFilter;

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);
        }
    }
}
```

Code Example 7-19 DateLaterThanPropertiesFilter.java File (Continued)

```

    }

    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 /etc/opt/SUNWps/desktop/classes -classpath
portal-server-install-root/SUNWps/sdk/desktop/desktopsdk.jar:identity-server-install-root/S
UNWam/lib/servlet.jar DateLaterThanPropertiesFilter.java

```

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

Code Example 7-20 contains the sample display profile XML fragment for the DateLaterThanPropertiesFilter.

Code Example 7-20 DateLaterThanPropertiesFilter.xml File

```

<?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>

```

Code Example 7-20 DateLaterThanPropertiesFilter.xml File

```

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

```

4. Load the display profile using the `dpadmin` command.

For example, to load the `DateLaterThanPropertiesFilter.xml` into the provider display profile, type:

```
portal-server-install-root/SUNWps/bin/dpadmin modify -m -p channelName -u dn_admin
-w password -d distinguishedname DateLaterThanPropertiesFilter.xml
```

Alternatively, you can also use the administration console to add and/or modify conditional properties. See *Portal Server Administration Guide* 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:

- [Code Example 7-21](#) includes an implementation that will return “en 03/03/2003 nokia”

Code Example 7-21 Provider Implementation for `DateLaterThanPropertiesFilter`

```

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);

```

- [Code Example 7-22 on page 116](#) includes an implementation that will return “en nokia”

Code Example 7-22 Provider Implementation for DateLaterThanPropertiesFilter

```
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);
```

Extending the Leaf Providers

This chapter includes detailed instructions for:

- [Extending the JSPPProvider](#)
- [Extending the URLScrapperProvider](#)
- [Extending the XMLProvider](#)

Extending the JSPPProvider

This sections contains some sample extensions to the JSPPProvider.

Example 1

In the following example, the JSPPProvider `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 accomplish this:

1. Extend the JSPPProvider and develop the AddressBookProvider class.

The AddressBookProvider class overrides the JSPPProvider'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.

Code Example 8-1 AddressBookProvider.java File

```
package custom;

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

Code Example 8-1 AddressBookProvider.java File (Continued)

```
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 /etc/opt/SUNWps/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 (see [Code Example 8-2](#)) and the display profile fragment for the AddressBookProvider's channel is saved in AddressBookChannel.xml file (see [Code Example 8-3 on page 119](#)).

Code Example 8-2 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"/>
  </Properties>
</Provider>
```


Code Example 8-2 AddressBookProvider.xml File (Continued)

```

    <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 [Code Example 8-3](#), note that the AddressBookProvider channel definition is embedded in an existing table container, JSPTableContainer.

Code Example 8-3 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 `dpadmin` command to upload the display profile fragments for this provider.

For the `AddressBookProvider`, use the `dpadmin` command to upload the `AddressBookProvider.xml` file and `AddressBookChannel.xml` file XML fragments in the display profile. That is, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_amadmin -w password -d
distinguishedname AddressBookProvider.xml
```

```
portal-server-install-root/SUNWps/bin/dpadmin modify -m -u dn_amadmin -w password -d
distinguishedname AddressBookChannel.xml
```

6. Develop the JSP file for the provider.

For the sample `AddressBookProvider`, the JSP in [Code Example 8-4 on page 120](#) 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 [Code Example 8-4](#).

Code Example 8-4 `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 `/etc/opt/SUNWps/desktop/desktoptype`. For this example, create an `AddressBook` directory under `/etc/opt/SUNWps/desktop/desktoptype` directory.

8. Copy all the JSP files over to the newly created directory (in [Step 7](#)). For example, type:

```
cp addressBook.jsp /etc/opt/SUNWps/desktop/desktoptype/AddressBook
```

9. Access the AddressBook channel inside the JSPTableContainer. To access, log in to the Desktop and type the following URL in your browser:

`http://hostname:port/portal/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:

1. Extend the `JSPProvider` and develop the `SimpleUserInfoProvider` class.

The `SimpleUserInfoProvider` class overrides the `JSPProvider processEdit()` method as shown in [Code Example 8-5 on page 121](#).

Code Example 8-5 `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 `/etc/opt/SUNWps/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 /etc/opt/SUNWps/desktop/classes -classpath
portal-server-install-root/SUNWps/sdk/desktop/desktopsdk.jar:identity-server-install-root/SUNWam/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 (see [Code Example 8-6 on page 122](#)) and the display profile fragment for the SimpleUserInfoProvider's channel is saved in SimpleUIChannel.xml file (see [Code Example 8-7 on page 122](#)).

Code Example 8-6 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 [Code Example 8-7](#), note that the SimpleUserInfoProvider channel definition is embedded inside JSPTableContainer thus making the channel only visible via the container.

Code Example 8-7 SimpleUIChannel.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">
```

Code Example 8-7 SimpleUIPChannel.xml File (Continued)

```

        <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 `dpadmin` command to upload the display profile fragments for this provider.

For the `SimpleUserInfoProvider`, use the `dpadmin` command to upload the `SimpleUIProvider.xml` file and `SimpleUIPChannel.xml` file XML fragments in the display profile. That is, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_amadmin -w password -d
distinguishedname SimpleUIProvider.xml
```

```
portal-server-install-root/SUNWps/bin/dpadmin modify -m -u dn_amadmin -w password -d
distinguishedname SimpleUIPChannel.xml
```

5. Create a channel directory under the template root directory.

By default, the template root directory is `/etc/opt/SUNWps/desktop/desktoptype`. For this example, create an `SimpleUserInfo` directory under `/etc/opt/SUNWps/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 `/etc/opt/SUNWps/desktop/desktoptype/SimpleUserInfo` directory.

7. Access the SimpleUserInfo channel inside the JSPTableContainer. To access, log in to the Desktop and type the following URL in your browser:

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

Extending the URLScrapperProvider

The following sample extends the URLScrapperProvider 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.

1. Extend the URLScrapperProvider and develop the custom class file.

The CustomURLScrapperProvider class file overrides the URLScrapperProvider `getURL()` method as shown in [Code Example 8-8](#). The CustomURLScrapperProvider gets the URL property for the provider. This is the URL from where the contents are fetched.

Code Example 8-8 CustomURLScrapperProvider.java File

```
package custom;

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

import com.sun.portal.providers.urlscrapper.URLScrapperProvider;
import com.sun.portal.providers.ProviderException;
import com.sun.portal.providers.context.ProviderContextException;

public class CustomURLScrapperProvider
extends URLScrapperProvider {
    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 `/etc/opt/SUNWps/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 `CustomURLScrapperProvider.java` file, type:

```
javac -d /etc/opt/SUNWps/desktop/classes -classpath
portal-server-install-root/SUNWps/sdk/desktop/desktopsdk.jar:identity-server-install-root/SUNWam/lib/servlet.jar CustomURLScrapperProvider.java
```

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

The display profile fragment for the `CustomURLScrapperProvider`'s provider is saved in `CustomURLSProvider.xml` file (see [Code Example 8-9](#)) and the display profile fragment for the `CustomURLScrapperProvider`'s channel is saved in `CustomURLSChannel.xml` file (see [Code Example 8-10 on page 126](#)).

Code Example 8-9 CustomURLSProvider.xml File

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

<Provider name="CustomURLScrapperProvider" class="custom.CustomURLScrapperProvider">
  <Properties>
    <String name="title" value=" Custom UrlScrapper Channel"/>
    <String name="description" value="This channel displays the urls at random."/>
    <Boolean name="isEditable" value="false" advanced="true"/>
    <String name="urlScrapperRulesetID" 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="fontFacel" 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>
```

Code Example 8-10 CustomURLSChannel.xml File

```

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

<Channel name="CustomURLScrapper" provider="CustomURLScrapperProvider">
  <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 `dpadmin` command to upload the display profile fragments for this provider.

For the `CustomURLScrapperProvider`, use the `dpadmin` command to upload the `CustomURLSProvider.xml` and `CustomURLSChannel.xml` file XML fragments in the display profile. That is, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_admin -w password -d
distinguishedname CustomURLSProvider.xml
```

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_admin -w password -d
distinguishedname CustomURLSChannel.xml
```

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

The `CustomURLScrapperProvider` channel will be displayed in `JSPTTableContainer`. To add the channel in the `JSPTTableContainer` from the administration console, follow instructions in *Portal Server Administration Guide*. To add the channel in the `JSPTTableContainer` manually:

- a. Add the `CustomURLScrapperProvider` channel XML fragment (in the `CustomURLSContainer.xml` file) for the `JSPTTableContainerProvider`.

The `CustomURLScrapperProvider` container XML fragment is shown in [Code Example 8-11](#).

Code Example 8-11 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="CustomURLScrapper" />
    </Collection>
    <Collection name="channelsRow">
      <String name="CustomURLScrapper" value="3" />
    </Collection>
  </Properties>
  <Available>
    <Reference value="CustomURLSscraper" />
  </Available>
  <Selected>
    <Reference value="CustomURLScrapper" />
  </Selected>
  <Channels>
  </Channels>
</Container>

```

- b. Use the `dpadmin modify` 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.

```

portal-server-install-root/SUNWps/bin/dpadmin modify -m -u
dn_amadmin -w password -d distinguishedname CustomURLSContainer.xml

```

6. Change directories to `/etc/opt/SUNWps/desktop/classes` and copy the properties file for the provider into this directory.

For example, for the CustomURLScrapperProvider:

- a. Change directories to `WebContainer/portal/web-apps/WEB-INF/classes`.
- b. Type `cp URLScrapperProvider.properties /etc/opt/SUNWps/desktop/classes/CustomURLScrapperProvider.properties`.

7. Access the CustomURLScrapperProvider channel inside the JSPTableContainer. To access, log in to the Desktop and type the following URL in your browser:

```

http://hostname:port/portal/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 accomplish this:

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

The CustomXMLProvider class file (see [Code Example 8-12](#)) extends the XMLProvider and includes only the overriding `getXML()` method.

Code Example 8-12 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
        }
    }
}
```

Code Example 8-12 CustomXMLProvider.java File (Continued)

```

    }
    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 `/etc/opt/SUNWps/desktop/classes`. To compile the `CustomXMLProvider.java` file, type:

```
javac -d /etc/opt/SUNWps/desktop/classes -classpath
portal-server-install-root/SUNWps/sdk/desktop/desktopsdk.jar:identity-server-install-root/SUNWam/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 (see [Code Example 8-13](#)) and the display profile fragment for the `CustomXMLProvider`'s channel is saved in `CustomXMLChannel.xml` file (see [Code Example 8-14 on page 130](#)).

Code Example 8-13 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="urlScrapperRulesetID" value="default_ruleset"/>
    <Boolean name="cookiesToForwardAll" value="true"/>
  </Properties>
</Provider>

```

Code Example 8-13 CustomXMLProvider.xml File (Continued)

```

        <Collection name="cookiesToForwardList">
        </Collection>
    </Properties>
</Provider>

```

Code Example 8-14 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 `dpadmin` command to upload the display profile fragments for this provider.

For the CustomXMLProvider, use the `dpadmin` command to upload the CustomXMLProvider.xml and CustomXMLChannel.xml file XML fragments in the display profile. That is, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_admin -w password -d
distinguishedname CustomXMLProvider.xml
```

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_admin -w password -d
distinguishedname CustomXMLChannel.xml
```

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

The CustomXMLProvider channel will be displayed in JSPTTableContainer. To add the channel in the JSPTTableContainer from the administration console, follow instructions in the *Portal Server Administration Guide*. To add the channel in the JSPTTableContainer manually:

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

The CustomXMLProvider container fragment is shown in [Code Example 8-15](#).

Code Example 8-15 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 `dpadmin modify` 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.

```
portal-server-install-root/SUNWps/bin/dpadmin modify -m -u dn_amadmin
-w password -d distinguishedname CustomXMLContainer.xml
```

6. Change directories to `/etc/opt/SUNWps/desktop/classes` and copy the properties file for the provider into this directory.

For example, for the CustomXMLProvider:

- a. Change directories to `WebContainer/portal/web-apps/WEB-INF/classes`.
- b. Type `cp XMLProvider.properties /etc/opt/SUNWps/desktop/classes/CustomXMLProvider.properties`.

7. Access the CustomXMLProvider channel inside the JSPTableContainer. To access, log in to the Desktop and type the following URL in your browser:

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

Extending the Container Providers

This chapter contains instructions for:

- [Extending the JSPSingleContainerProvider](#)
- [Extending the JSPTableContainerProvider](#)
- [Extending the JSPTabContainerProvider](#)

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`.

1. Extend the `JSPSingleContainerProvider` and develop the `CustomSingleContainerProvider` class file.

The `CustomSingleContainerProvider` class overrides the `JSPSingleContainerProvider.getSelectedChannel()` method as shown in [Code Example 9-1](#).

Code Example 9-1 `CustomSingleContainerProvider.java` File

```
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;
```

Code Example 9-1 CustomSingleContainerProvider.java File (Continued)

```

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 `/etc/opt/SUNWps/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 /etc/opt/SUNWps/desktop/classes -classpath
portal-server-install-root/SUNWps/sdk/desktop/desktopsdk.jar :identity-server-install-root/SUNWam/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 [Code Example 9-2](#)) and the display profile fragment for the `CustomSingleContainerProvider`'s channel is saved in `CustomSCChannel.xml` file (see [Code Example 9-3 on page 135](#)).

Code Example 9-2 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>

```


Code Example 9-2 CustomSCProvider.xml File (Continued)

```

    <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>

```

Code Example 9-3 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 `dpadmin` command to upload the display profile fragments for this provider.

For the `CustomSingleContainerProvider`, use the `dpadmin` command to upload the `CustomSCProvider.xml` file and `CustomSCChannel.xml` file XML fragments into the display profile. That is, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_amadmin -w password -d
distinguishedname CustomSCProvider.xml
```

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_amadmin -w password -d
distinguishedname CustomSCChannel.xml
```

5. Create a new directory for the provider in `/etc/opt/SUNWps/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 `/etc/opt/SUNWps/desktop/desktoptype` directory.

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

For the `CustomSingleContainerProvider`, copy files from `/etc/opt/SUNWps/desktop/default/JSPSingleContainerProvider` to the `/etc/opt/SUNWps/desktop/desktoptype/CustomSingleContainer` directory.

- a. Modify `single.jsp` file in the `/etc/opt/SUNWps/desktop/desktoptype/CustomSingleContainer` directory as shown in [Code Example 9-4](#).

Code Example 9-4 Modifications to `single.jsp` File

```
<!--
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>
```

Code Example 9-4 Modifications to `single.jsp` File (Continued)

```

<br>

<%
CustomSingleContainerProvider csp =
(CustomSingleContainerProvider)pageContext.getAttribute("JSPPProvider");
%>

<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 administration console.

7. Access the `CustomSingleContainerProvider` from your browser. To access, log in to the Desktop and type the following URL in your browser:

`http://hostname:port/portal/dt?provider=CustomSingleContainer`

Extending the JSPTableContainerProvider

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

Example 1

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

1. Extend JSPTableContainerProvider and develop the provider class file.

For CustomTableContainerProvider, the CustomTableContainerProvider class is implemented by extending the JSPTableContainerProvider class as shown in [Code Example 9-5](#).

Code Example 9-5 CustomTableContainerProvider.java File

```

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 newList = new ArrayList();
        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 `/etc/opt/SUNWps/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 /etc/opt/SUNWps/desktop/classes -classpath
portal-server-install-root/SUNWps/sdk/desktop/desktopsdk.jar:identity-server-install-root/SUNWam/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 (see [Code Example 9-6](#)) and the display profile fragment for the `CustomTableContainerProvider`'s channel is saved in `CustomTCChannel.xml` file (see [Code Example 9-7 on page 141](#)).

Code Example 9-6 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="fontFacel" value="Sans-serif"/>
    <String name="Desktop-fontFacel" 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">
```

Code Example 9-6 CustomTCProvider.xml File (Continued)

```

        <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="SampleURLScrapper"/>
    </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>

```

Code Example 9-7 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-fontFacel" 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"/>
  </Available>
</Container>

```

Code Example 9-7 CustomTCChannel.xml File (Continued)

```

    <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 `dpadmin` command to upload the display profile fragments for this provider.

For `CustomTableContainerProvider`, use the `dpadmin` command to upload the `CustomTCProvider.xml` file and `CustomTCChannel.xml` file fragments in the display profile. That is, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_admin -w password -d
distinguishedname CustomTCProvider.xml
```

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_admin -w
password -d distinguishedname CustomTCChannel.xml
```

5. Create a new directory for the provider in `/etc/opt/SUNWps/desktop/desktoptype` directory. The directory for the provider is typically named after the channel.

For the sample `CustomTableContainerProvider`, create a directory called `CustomTableContainer` in `/etc/opt/SUNWps/desktop/desktoptype`.

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

For `CustomTableContainerProvider`, copy files from `/etc/opt/SUNWps/desktop/default/JSPTableContainerProvider` to the `/etc/opt/SUNWps/desktop/desktoptype/CustomTableContainer` directory.

7. Copy the resource file for the provider into `/etc/opt/SUNWps/desktop/classes` directory.

For example, for the `CustomTableContainerProvider`:

- a. Change directories to `portal-server-install-root/SUNWps/web-src/WEB-INF/classes` directory.
- b. Type `cp JSPTTableContainerProvider.properties /etc/opt/SUNWps/desktop/classes/CustomTableContainerProvider.properties`.

8. Access `CustomTableContainerProvider` from your browser. To access, log in to the Desktop and type the following URL in your browser:

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

Example 2

[Figure 9-1](#) shows a sample layout for a `Thin_Wide_Thin_Span` column. The table Desktop layout in [Figure 9-1](#) has three columns with the `full_bottom` spanning two (left and center) columns.

Figure 9-1 Sample Table Column Span

To achieve the layout shown in [Figure 9-1](#):

1. Extend `JSPTTableContainerProvider` and develop the `SpanTableContainerProvider` class file.

The `SpanTableContainerProvider` class is implemented by extending the `JSPTTableContainerProvider` class as shown in [Code Example 9-8 on page 144](#).

Code Example 9-8 SpanTableContainerProvider.java File

```

package custom;

import com.sun.portal.providers.containers.jsp.table.JSPTTableContainerProvider;
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 JSPTTableContainerProvider {
    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 `/etc/opt/SUNWps/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 /etc/opt/SUNWps/desktop/classes -classpath
portal-server-install-root/SUNWps/sdk/desktop/desktopsdk.jar:identity-server-install-root/S
UNWam/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 (see [Code Example 9-9 on page 145](#)) and the display profile fragment for SpanTableContainerProvider's channel is saved in SpanTCChannel.xml file (see [Code Example 9-10 on page 146](#)).

Code Example 9-9 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="fontFacel" value="Sans-serif"/>
    <String name="Desktop-fontFacel" 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"/>
    </Collection>
  </Properties>
</Provider>
```

Code Example 9-9 SpanTCPProvider.xml File (Continued)

```

        <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>

```

Code Example 9-10 SpanTCCChannel.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-fontFacel" value="Sans-serif"/>
    <Collection name="categories">
      <String value="Personal Channels"/>
      <String value="Sample Channels"/>
      <String value="News Channels"/>
    </Collection>
  </Properties>
</Container>

```

Code Example 9-10 SpanTCCChannel.xml File (Continued)

```

</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"/>

```

Code Example 9-10 SpanTCChannel.xml File (Continued)

```

        <Reference value="SampleRSS" />
        <Reference value="SampleXML" />
    </Selected>
    <Channels>
    </Channels>
</Container>

```

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

For `SpanTableContainerProvider`, use the `dpadmin` command to upload the `CustomTCProvider.xml` file and `CustomTCChannel.xml` file fragments in the display profile. That is, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_amadmin -w password -d
distinguishedname SpanTCProvider.xml
```

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_amadmin -w password -d
distinguishedname SpanTCChannel.xml
```

5. Create a new directory for the provider in `/etc/opt/SUNWps/desktop/desktoptype` directory. The directory for the provider is typically named after the channel.

For the sample `SpanTableContainerProvider`, create a directory called `SpanTableContainerChannel` in `/etc/opt/SUNWps/desktop/desktoptype` directory.

6. Copy files from `/etc/opt/SUNWps/desktop/default/JSPTableContainerProvider` to the newly created directory.
7. Modify the JSP files (in `/etc/opt/SUNWps/desktop/desktoptype/SpanTableContainerChannel`) as shown in Step a, Step b, and Step c (modifications to the file are shown in bold):
 - a. Add the following to `toptable.jsp`:

Code Example 9-11 Changes to toptable.jsp File

```

<!-- BEGIN FULL TOP CHANNELS -->
<dtable: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 -->
<dtable: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 following to `tablecolumn.jsp`:

Code Example 9-12 Sample `tablecolumn.jsp` File

```

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

```

- c. Add the following to `tabletopbottom.jsp`:

Code Example 9-13 Sample `tabletopbottom.jsp`

```
<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 `WebContainer/portal/web-apps/WEB-INF/classes/JSPTableContainerProvider.properties` to `/etc/opt/SUNWps/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 layout described in [Figure 9-1 on page 143](#) to be displayed.
10. Access `SpanTableContainerProvider` from your browser. To access, type the following URL in your browser:

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

Example 3

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

Customizing the Row Layout

1. Create a new custom table container provider by extending `JSPTableContainerProvider` as shown in [Code Example 9-14](#).

Override the `getLayout()`, `setUpColumns()`, and `getWidths()` methods as shown in [Code Example 9-14](#) 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.

Code Example 9-14 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);
            }
        }
    }
}
```

Code Example 9-14 CustomTCProvider.java File (Continued)

```

        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 columnMap = new HashMap();

    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 `/etc/opt/SUNWps/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 /etc/opt/SUNWps/desktop/classes -classpath
portal-server-install-root/SUNWps/sdk/desktop/desktopsdk.jar:identity-server-install-root/SUNWam/lib/servlet.jar CustomTCPProvider.java
```

3. Develop the display profile XML fragment for this provider and this provider's channel.
 - a. The display profile fragment for the CustomTCPProvider'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 CustomTCPProvider's provider.
4. Use the `dpadmin` command to upload the display profile fragments for this provider.

For CustomTCPProvider, use the `dpadmin` command to upload the CustomTCPProvider.xml file and CustomTCChannel.xml file fragments in the display profile. That is, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_amadmin -w password -d distinguishedname CustomTCPProvider.xml
```

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_amadmin -w password -d distinguishedname CustomTCChannel.xml
```

5. Create a new directory for the provider in `/etc/opt/SUNWps/desktop/desktopype`. The directory for the provider is typically named after the provider.

For the sample CustomTCPProvider, create a directory called CustomTCPProvider in `/etc/opt/SUNWps/desktop/desktopype`.

6. Copy files from `/etc/opt/SUNWps/desktop/default/JSPTTableContainerProvider` to the newly created directory.
7. Modify the HTML in `toptable.jsp` file in `/etc/opt/SUNWps/desktop/desktopype/CustomTCPProvider`.

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

8. Create a new directory for CustomLayoutContainer under `/etc/opt/SUNWps/desktop/desktopype`. For example, type:

```
mkdir /etc/opt/SUNWps/desktop/desktopype/CustomLayoutContainer
```

9. Copy files from `/etc/opt/SUNWps/desktop/default/JSPLayoutContainer` to `/etc/opt/SUNWps/desktop/desktoptype/CustomLayoutContainer`. For example, type:

```
cp /etc/opt/SUNWps/desktop/default/JSPLayoutContainer/*
   /etc/opt/SUNWps/desktop/desktoptype/CustomLayoutContainer
```

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

Modify the `layoutedit.jsp` file at

`/etc/opt/SUNWps/desktop/desktoptype/CustomLayoutContainer` as shown in [Code Example 9-15](#).

Code Example 9-15 Sample `layoutedit.jsp`

```
<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>
```

11. Create a new `layout5.jsp` which gets displayed on the layout page when the user selects layout 5.
12. Modify `layoutdoedit.jsp` to handle the processing of layout 5 that you have created.
13. Add the display profile for `CustomLayoutContainer` as shown in [Code Example 9-16](#).

Code Example 9-16 `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"/>
```

Code Example 9-16 CustomLayoutContainer.xml File (*Continued*)

```

        <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>

```

14. Upload the display profile XML fragment for CustomLayoutContainer using the dpadmin command.

For example, to upload CustomLayoutContainer.xml file fragments, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_amadmin -w password -d
distinguishedname CustomLayoutContainer.xml
```

15. Access CustomTCProvider from your browser. To access, type the following URL in your browser:

```
http://hostname:port/portal/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.

1. Extend the `JSPTableContainerProvider` and develop the `CustomJSPTableContainerProvider` class file.

The `CustomJSPTableContainerProvider` class file in [Code Example 9-17 on page 156](#) 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).

Code Example 9-17 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 `/etc/opt/SUNWps/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 `CustomJSPTTableContainerProvider.java` file, enter:

```
javac -d /etc/opt/SUNWps/desktop/classes -classpath
BaseDir/SUNWps/sdk/desktop/desktopsdk.jar:BaseDir/SUNWam/lib/servlet.jar
CustomJSPTTableContainerProvider.java
```

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

The display profile fragment for the `CustomJSPTTableContainerProvider`'s provider is saved in `CustomJSPTCProvider.xml` file (see [Code Example 9-18](#)) and the display profile fragment for the `CustomTableContainerProvider`'s channel is saved in `CustomJSPTCChannel.xml` file (see [Code Example 9-19](#) on page 159).

Code Example 9-18 CustomJSPTCProvider.xml File

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

<Provider name="CustomJSPTTableContainerProvider"
class="custom.CustomJSPTTableContainerProvider">
  <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="fontFacel" value="Sans-serif"/>
    <String name="Desktop-fontFacel" 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"/>
```

Code Example 9-18 CustomJSPTCProvider.xml File (Continued)

```

<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="SampleURLScrapper"/>
</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>

```


Code Example 9-19 CustomJSPTCChannel.xml File

```

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

<Container name="CustomJSPTTableContainerChannel"
provider="CustomJSPTTableContainerProvider">
  <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-fontFacel" 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"/>

```

Code Example 9-19 CustomJSPTCChannel.xml File (Continued)

```

    <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 `dpadmin` command to upload the display profile fragments for this provider.

For `CustomJSPTTableContainerProvider`, use the `dpadmin` command to upload the `CustomTCProvider.xml` file and `CustomTCChannel.xml` file fragments in the display profile. That is, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_admin -w password -d
distinguishedname CustomJSPTCProvider.xml
```

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_admin -w password -d
distinguishedname CustomJSPTCChannel.xml
```

5. Create a new directory for the provider in `/etc/opt/SUNWps/desktop/desktoptype`. The directory for the provider is typically named after the provider.

For the sample `CustomJSPTTableContainerProvider`, create a directory called `CustomJSPTTableContainerChannel` in `/etc/opt/SUNWps/desktop/desktoptype`.

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

For CustomJSPTabContainerProvider, copy files from

`/etc/opt/SUNWps/desktop/default/JSPTabContainerProvider` to the `/etc/opt/SUNWps/desktop/desktoptype/CustomJSPTabContainerChannel` directory.

7. Copy the resource file for the provider into

`/etc/opt/SUNWps/desktop/classes` directory.

For example, for the CustomJSPTabContainerProvider:

- a. Change directories to `WebContainer/portal/web-apps/WEB-INF/classes` directory.

- b. Type `cp JSPTabContainerProvider.properties`

`/etc/opt/SUNWps/desktop/classes/CustomJSPTabContainerProvider.properties.`

8. Access CustomJSPTabContainerProvider from your browser. To access, log in to the Desktop and type the following URL in your browser:

`http://hostname:port/portal/dt?provider=CustomJSPTabContainerChannel`

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 accomplish this:

1. Extend the JSPTabContainerProvider and develop the CustomTabContainerProvider class file.

The CustomTabContainerProvider class file in [Code Example 9-20](#) includes a `getTabTopics()` method that returns only the predefined tabs from the list of available tabs.

Code Example 9-20 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 availtabs = new ArrayList();
        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 `/etc/opt/SUNWps/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 /etc/opt/SUNWps/desktop/classes -classpath
portal-server-install-root/SUNWps/sdk/desktop/desktopsdk.jar:identity-server-install-root/SUNWam/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` (see [Code Example 9-21](#)) and `CustomTabCChannel.xml` (see [Code Example 9-22 on page 164](#)) files respectively.

Code Example 9-21 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="fontFacel" 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>

```

Code Example 9-22 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"/>

```

Code Example 9-22 CustomTabCChannel.xml File (Continued)

```

        <String name="desc" value="Samples 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">

```

Code Example 9-22 CustomTabCChannel.xml File (Continued)

```

        <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 `dpadmin` command to upload the display profile fragments for this provider.

For `CustomTabContainerProvider`, use the `dpadmin` command to upload the `CustomTabCProvider.xml` file fragments in the display profile. That is, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_admin -w password -d
distinguishedname CustomTabCProvider.xml
```

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_admin -w password -d
distinguishedname CustomTabCChannel.xml
```

5. Create a new directory called `CustomTabContainerProvider` under `/etc/opt/SUNWps/desktop/desktoptype`.
6. Copy all the required JSP files from `/etc/opt/SUNWps/desktop/sampleportal/JSPTabContainerProvider` to `/etc/opt/SUNWps/desktop/desktoptype/CustomTabContainerProvider`.
7. Modify the `makeNewTab.jsp` file in `/etc/opt/SUNWps/desktop/desktoptype/CustomTabContainerProvider` to call `getTabTopics()`.

That is, replace the `makeNewTab.jsp` file with the `makeNewTab.jsp` file in [Code Example 9-23](#).

Code Example 9-23 Sample Modifications to `makeNewTab.jsp` File

```

<%--
Copyright 2001 Sun Microsystems, Inc. All rights reserved.
PROPRIETARY/CONFIDENTIAL. Use of this product is subject to license terms.

```


Code Example 9-23 Sample Modifications to makeNewTab.jsp File (Continued)

```

--%>
<%-- makeNewTab.jsp --%><%@ page import="custom.CustomTabContainerProvider" %>
<table border=0 cellspacing=2 cellpadding=2 width="50%">
  <tr>
    <td align="right"><font face="<%=fontFacel%" size="+0"><b>Tab
Name:</b></font></td>
    <td><font face="<%=fontFacel%" size="+0"><input type="text"
name="JSPTabContainer.tabName" size="30"></font></td>
  </tr>
  <tr>
    <td align="right"><font face="<%=fontFacel%" size="+0"><b>Tab
Description:</b></font></td>
    <td><font face="<%=fontFacel%" size="+0"><input type="text"
name="JSPTabContainer.tabDesc" size="30"></font></td>
  </tr>
  <tr>
    <td align="right"><font face="<%=fontFacel%" 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">

```

Code Example 9-23 Sample Modifications to makeNewTab.jsp File (*Continued*)

```
        </td>
    </tr>
</table>
```

8. Copy the resource file for the provider into
/etc/opt/SUNWps/desktop/classes directory.

For example, for the CustomJSPTabContainerProvider:

- a. Change directories to *WebContainer*/portal/web-apps/WEB-INF/classes directory.
 - b. Type `cp JSPTabContainerProvider.properties /etc/opt/SUNWps/desktop/classes/CustomTabContainerProvider.properties`.
9. Access CustomTabContainerProvider from your browser. To access, log in to the Desktop and type the following URL in your browser:

`http://hostname:port/portal/dt?provider=CustomTabContainer`

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.

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 [Code Example 10-1](#) for the sample RouterContainerProvider.java file:

Code Example 10-1 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;
```

Code Example 10-1 RouterContainerProvider.java File (Continued)

```
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 selectedChannels = new ArrayList();
        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 /etc/opt/SUNWps/desktop/classes -classpath
portal-server-install-root/SUNWps/sdk/desktop/desktopsdk.jar:identity-server-install-root/S
UNWam/lib/servlet.jar RouterContainerProvider.java
```

3. Define the ContainerProvider provider and container channel XML fragments in the display profile.

The sample RouterContainerProvider provider XML fragment in the RCPProvider.xml file is shown [Code Example 10-2](#).

Code Example 10-2 RCPProvider.xml File

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

Code Example 10-2 RCProvider.xml File (Continued)

```
<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="fontFacel" 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 in the RCChannel.xml file is shown in [Code Example 10-3](#).

Code Example 10-3 RCChannel.xml File

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

  <Container name="JSRouterContainer" 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="JSPTTableContainer"/>
  </Available>
```

Code Example 10-3 RCChannel.xml File (Continued)

```
<Selected>
  <Reference value="JSPTabContainer"/>
</Selected>

<Channels>
</Channels>
</Container>
```

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

For `RouterContainerProvider`, use the `dpadmin` command to upload the `RCProvider.xml` file and `RCChannel.xml` file fragments in to the display profile. That is, type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_admin -w password -d
distinguishedname RCProvider.xml
```

```
portal-server-install-root/SUNWps/bin/dpadmin add -u dn_admin -w password -d
distinguishedname RCChannel.xml
```

5. Develop the templates or JSP files for the `ContainerProvider`.

The sample `RouterContainerProvider` requires the following JSP files:

- `router.jsp` (see [Code Example 10-4 on page 172](#)) - This file includes the logic for displaying the tab or the table Desktop based on the user's selection.
- `roureredit.jsp` ([Code Example 10-5 on page 173](#)) - This file has the logic for displaying the form where the user can select a tab or table Desktop.

Code Example 10-4 Sample `router.jsp` File

```
<%-- router.jsp --%>

<%@ page import="com.sample.providers.containers.router.RouterContainerProvider"%>

<%
RouterContainerProvider rcp =
(RouterContainerProvider)pageContext.getAttribute("JSPPProvider");
%>
<%= rcp.getContainerProviderContext().getContent(request, response, null,
rcp.getSelectedChannel())%>
```

Code Example 10-5 Sample routeredit.jsp File

```
<!-- 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="dtcp" %>
<%@ 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="fontFace" id="fontFace"/>
        <jx:declare id="fontFace" type="java.lang.String"/>
        <%
            String fontFace1 = (String)pageContext.getAttribute("fontFace1",
PageContext.REQUEST_SCOPE);
        %>
    </dtpc:providerContext>
</dt:obtainContainer>

<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>
        <tr>
            <td>&nbsp;</td>
        </tr>
        <tr>
            <td align=center>
                <SELECT NAME="desktop">
```

Code Example 10-5 Sample routeredit.jsp File (Continued)

```
<jx:forEach var="channel" items="$availableChannel">
<dtcpc:obtainChannelFromContainer channel="$channel">
<jx:declare id="channel" type="java.lang.String"/>
<dt:getTitle id="title" scope="request" silentException="true"/>
<%
    String title = (String)pageContext.getAttribute("title",
PageContext.REQUEST_SCOPE);
    String selected = "";
    if (selectedChannel.equals(channel)) {
        selected = "selected";
    }
%>
<OPTION <%=selected%> VALUE="<%=channel%>"><%=title%></OPTION>
</dtcpc:obtainChannelFromContainer>
</jx:forEach>
</SELECT>
</td>
</tr>
</table>
</CENTER>

<br>
<font size=+0 face="<%=fontFacel%>">
    <input type=SUBMIT name="Submit" value="Finished" CLASS="button">
    <input type=BUTTON onClick="location='<dtcpc:getDesktopURL/>' " value="Cancel "
CLASS="button">
</font>
<br>
<p>
</form>
<br>

<jsp:include page="menubar.jsp" flush="true"/>
<%@ include file="footer.html" %>

</dtcpc:providerContext>
</dt:obtainContainer>
```

6. Make a directory for the files in `/etc/opt/SUNWps/desktop/desktoptype` directory and copy the files over to the newly created directory.

For the sample RouterContainerProvider, type:

```
mkdir /etc/opt/SUNWps/desktop/desktoptype/JSPRouterContainer
cp router.jsp /etc/opt/SUNWps/desktop/desktoptype/JSPRouterContainer
cp routeredit.jsp /etc/opt/SUNWps/desktop/desktoptype/JSPRouterContainer
```


The RouterContainerProvider also requires header.jsp to display the header on routeredit page and menubar.jsp to display the menubar on the routeredit page. Copy these files from /etc/opt/SUNWps/desktop/default/JSPSingleContainerProvider directory.

```
cp /etc/opt/SUNWps/desktop/sampleportal/JSPSingleContainer/header.jsp
/etc/opt/SUNWps/desktop/desktoptype/JSPRouterContainer
cp /etc/opt/SUNWps/desktop/sampleportal/JSPSingleContainer/menubar.jsp
/etc/opt/SUNWps/desktop/desktoptype/JSPRouterContainer
```

7. Modify the header.jsp file in

/etc/opt/SUNWps/desktop/sampleportal/JSPTabContainer to add the text from tabheader.jsp file (as shown in Code Example 10-6).

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 Code Example 10-6) file. Add this HTML code to

/etc/opt/SUNWps/desktop/sampleportal/JSPTabContainer/header.jsp and /etc/opt/SUNWps/desktop/sampleportal/JSPTableContainerProvider/header.jsp files.

Code Example 10-6 Sample tabheader.jsp File

```
<!--
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')"> </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>
    <br>
```

8. Modify the header . jsp file in

/etc/opt/SUNWps/desktop/default/JSPTableContainerProvider to add the text from tableheader . jsp file (as shown in [Code Example 10-7](#)).

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 [Code Example 10-7 on page 176](#)) file.

Code Example 10-7 Sample tableheader . jsp 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"></td>
      </TR>
      <TR>
        <td><a href="dt?action=edit&provider=JSRouterContainer"
onMouseOver="over('banner_home')" onMouseOut="out('banner_home')"> </a></td>
        <td><a href="dt?action=edit&provider=JSRouterContainer"
onMouseOver="over('banner_home')" onMouseOut="out('banner_home')" class="noUnderline">
<span class="banner-links">Desktop Preference</span></a> </td>
      </TR>
    </TABLE>
    <br>
```

9. Open a browser and log in to the administration console.

The URL to access the administration console is
`http://hostname:port/amconsole`.

10. Change the default channel name to your ContainerProvider.

To specify the default channel name, select Services from the View pull-down menu for your organization and select Portal Desktop. For the sample RouterContainerProvider, change the default channel name to JSRouterContainer.

11. Save the settings and log out of the administration console.
12. Log in as the default user into the portal Desktop.
13. Select the Desktop Preference Link from the menubar.

The sample RouterContainerProvider provides the page to select Tabbed or Table Desktop.

Portlets

Chapter 11, “Overview of the Portlets”

Chapter 12, “Extending the GenericPortlet Abstract Class”

Chapter 13, “Converting Providers to Portlets”

Overview of the Portlets

This chapter contains the following sections:

- [What is a Portlet?](#)
- [Overview of Developing and Deploying Portlets](#)

What is a Portlet?

Portlet refers to pluggable web components that process requests and generate content within the context of a portal. In the Sun Java System Portal Server software, portlets are managed by a Portlet Container. Conceptually, portlets are equivalent to the providers.

Overview of Developing and Deploying Portlets

To develop and deploy portlets:

1. Develop the portlet class files.
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.

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.

This chapter contains the following sections:

- [Developing the Class File](#)
- [Compiling the Portlet](#)
- [Creating a Portlet Web Application](#)
- [Deploying the Application](#)
- [Creating Channels from the Deployed Portlets](#)

Developing the Class File

[Code Example 12-1](#) contains the class file for the sample `PrefPortlet`.

Code Example 12-1 PrefPortlet.java File

```
package examples;

import javax.portlet.GenericPortlet;
import javax.portlet.ActionRequest;
import javax.portlet.RenderRequest;
import javax.portlet.ActionResponse;
```

Code Example 12-1 PrefPortlet.java File (Continued)

```

import javax.portlet.RenderResponse;
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 {
        // prompt the user for the salutation.
        PortletURL actionURL = response.createActionURL();
        response.setContentType(request.getResponseContentType());
        PrintWriter writer = response.getWriter();
        writer.write("<form method='post' action='" + actionURL.toString();
        writer.write("'><TABLE WIDTH='100%'><TR><TD ALIGN='RIGHT'
VALIGN='TOP'>salutation:</TD><TD ALIGN='LEFT'><INPUT TYPE='TEXT'
NAME='SALUTATION'></TD></TR><TR><TD ALIGN='RIGHT'>&nbsp;</TD><TD ALIGN='LEFT'><INPUT
TYPE='SUBMIT' NAME='SUB1' VALUE='Submit'></TD></TR></TABLE></form>");
    }

    public void doHelp(RenderRequest request, RenderResponse response) throws
PortletException {

```

Code Example 12-1 PrefPortlet.java File (Continued)

```

        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

When compiling a portlet, include the classpath. Ensure that `portlet.jar` is in your classpath when compiling the portlet. For example, to compile, type:

```
javac -classpath portal-server-install-root/SUNWps/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 `pdeploy` 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 WAR file. The `portlet.xml` file must be placed into the same location as the `web.xml` file.

To create a portlet web application:

1. Create a complete `portlet.xml` file that includes the declaration for all the portlets.

[Code Example 12-2](#) contains the `portlet.xml` file that includes the declaration for the sample portlet. In [Code Example 12-2](#), the 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.

Code Example 12-2 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. [Code Example 12-3](#) contains the web.xml file for the sample portlet.

Code Example 12-3 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 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.

```
pdeploy deploy -u uid -w amAdminPassword -g|-d distinguishedName -p
ContainerAdminPassword examples.war
```

For more information on the `pdeploy` command, see *Portal Server Administration Guide*.

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 *Portal Server Administration Guide* for more information.

Converting Providers to Portlets

This chapter includes instructions for converting the providers to JSR168 compatible portlets using a sample set of instructions for converting the HelloWorldProviderPA in [Chapter 7, “Extending the Base Providers”](#) into a portlet called HelloWorldPortlet.

This chapter contains the following sections:

- [Developing the Class File](#)
- [Compiling the Portlet](#)
- [Creating a Portlet Web Application](#)
- [Deploying the Application](#)
- [Creating Channels from the Deployed Portlets](#)

Developing the Class File

[Code Example 13-1](#) contains the HelloWorldPortlet class file. The HelloWorldPortlet, similar to the HelloWorldProviderPA, prints “Hello World!” on the channel.

Code Example 13-1 HelloWorldPortlet.java File

```
package examples;

import java.io.IOException;
import java.io.PrintWriter;

import javax.portlet.GenericPortlet;
import javax.portlet.PortletException;
import javax.portlet.RenderRequest;
```

Code Example 13-1 HelloWorldPortlet.java File *(Continued)*

```
import javax.portlet.RenderResponse;

public class HelloWorldPortlet extends GenericPortlet {
    public void doView(RenderRequest request,RenderResponse response)
        throws PortletException, IOException {
        response.setContentType(request.getResponseContentType());
        PrintWriter writer = response.getWriter();
        writer.write("Hello World");
    }
}
```

Mapping ProviderAdapter to GenericPortlet

When developing the class file, use the following table comparing GenericPortlet to ProviderAdapter:

Table 13-1 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 response)
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 ()

Table 13-1 Mapping ProviderAdapter to GenericPortlet (*Continued*)

ProviderAdapter	GenericPortlet
	<code>public String getInitParameter(java.lang.String name)</code>
	<code>public java.util.Enumeration getInitParameterNames()</code>
	<code>public void destroy ()</code>
<code>public int getEditType()</code>	
<code>public int getWidth()</code>	
<code>public boolean isEditable()</code>	
<code>public boolean isPresentable()</code>	
<code>public ProviderContext getProviderContext()</code>	

Sample Code Fragments for Provider to Portlet Conversion

This section includes some sample code fragments for Provider to Portlet conversion.

Example 1

The

```
ProviderContext.getStringAttribute("firstname");
```

maps to:

```
Map ui = (Map)renderRequest.getAttribute("javax.portlet.userinfo");
String firstname = (String)ui.get("firstname");
```

Example 2

The

```
ProviderContext.getSessionProperty()/setSessionProperty()
```

maps to (or can use)

```
PortletSession.getAttribute(), PortletSession.setAttribute()
```

Example 3

The

```
ProviderContext.getDesktopURL()
```

maps to (or can use)

```
PortletURL.RenderResponse.createRenderURL(),  
RenderResponse.createActionURL()
```

Dispatching to a JSP

```
PortletRequestDispatcher dispatcher =  
pContext.getRequestDispatcher("test.jsp");  
dispatcher.include(request, response);
```

Help

`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

`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

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 `/etc/opt/SUNWps/desktop/default` directory.

PortletPreferences

The

```
ProfileProviderAdapter.getStringProperty(key)
```

maps to

```
PortletPreferences.getValue(key, default)
```

The

```
ProfileProviderAdapter.getListProperty(key)
```

maps to

```
PortletPreferences.getValues(key, default[])
```

Compiling the Portlet

When compiling a portlet, include the classpath. Ensure that `portlet.jar` file is in your classpath when compiling the portlet. For example, to compile, type:

```
javac -classpath portal-server-install-root/SUNWps/lib/portlet.jar
HelloWorldPortlet.java
```

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 WAR file. The `portlet.xml` file must be placed into the same location as the `web.xml` file.

To create a portlet web application:

1. Create a `portlet.xml` file that includes the declaration for all the portlets.

Code Example 13-2 contains the `portlet.xml` file that includes the declaration for the sample `HelloWorldPortlet`.

Code Example 13-2 `portlet.xml` File for the `HelloWorldPortlet`

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:portlet="http://java.sun.com/xml/ns/portlet"
xsi:noNamespaceSchemaLocation="http://java.sun.com/xml/ns/portlet" version="1.0">
  <portlet>
    <portlet-name>HelloWorldPortlet</portlet-name>
    <portlet-class>custom.HelloWorldPortlet</portlet-class>
    <expiration-cache>0</expiration-cache>
    <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>VIEW</portlet-mode>
    </supports>
    <portlet-info>
      <title>HelloWorldPortlet</title>
    </portlet-info>
  </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 servlet definition can be incorporated in the `web.xml` file. **Code Example 13-3** contains the `web.xml` file for the sample `HelloWorldPortlet`.

Code Example 13-3 `web.xml` File for `HelloWorldPortlet`

```
<?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/HelloWorldPortlet.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.

```
pdeploy deploy -u uid -w adminpassword -g|-d distinguishedname -p
ContainerAdminPassword examples.war
```

For more information on the `pdeploy` command, see *Portal Server Administration Guide*.

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 *Portal Server Administration Guide* for more information.

Chapter 14, “Validating Registration Data”

Chapter 15, “Defining Custom Registration Validators”

Validating Registration Data

This chapter contains the following sections:

- [Overview](#)
- [Registering with the Producer](#)
- [Validating the Registration Data](#)

Overview

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 its capabilities and expectations through its 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), provided 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 objects 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>=<message>
```

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

Defining Custom Registration Validators

This chapter describes how to develop the registration validator. It contains the following sections:

- [Implementing the RegistrationValidator Interface](#)
- [Installing the Class File](#)
- [Customizing the Return Codes](#)
- [Configuring the Producer's Registration Validator](#)

Implementing the RegistrationValidator Interface

1. Implement the RegistrationValidator interface. For example, see [Code Example 15-1](#) for the DefaultRegistrationValidator class implementation:

Code Example 15-1 DefaultRegistrationValidator Class File

```
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;
```

Code Example 15-1 DefaultRegistrationValidator Class File (*Continued*)

```

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
portal-server-install-root/SUNWps/sdk/wsrp/wsrpsdk.jar:/portal-server-install-root/SUNWam/
lib/am_sdk.jar RegistrationValidatorImplementation.java
```

When compiling the class file, include the Identity Server SDK JAR file (*identity-server-install-root/SUNWam/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 `/opt/SUNWps/web-src`. If you have a standalone class file, copy the file into

portal-server-install-root/SUNWps/web-src/WEB-INF/classes directory; if you have a JAR file, copy it into *portal-server-install-root/SUNWps/web-src/WEB-INF/lib* directory.

2. Redeploy the web container. To redeploy, type *portal-server-install-root/SUNWps/bin/deploy redeploy -deploy_admin_password password*.

You can customize the return codes and then redeploy the web container. See [“For example, type `com.sun.portal.wsrp.producer.registration.validator.impl.RegistrationValidatorImplementation`.” on page 206](#) for more information.

Customizing the Return Codes

1. Log in to the Portal Server host and change directories to *portal-server-install-root/SUNWps/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.RegistrationValidatorImplementation -1=Missing Or Empty Registration Property*.

4. Redeploy the web application. To redeploy, use the `portal-server-install-root/SUNWps/bin/deploy redeploy -deploy_admin_password password` command.

Configuring the Producer's Registration Validator

To administer the producer:

1. Log in to the administration console and navigate to the producer's administration page.

To navigate to the producer's administration page, you must select Services from the View pull-down menu → WSRP Producers → Edit 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 wish 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.

Search Engine Robot

Chapter 16, “Search Engine Robot Overview”

Chapter 17, “Robot Completion Scripts”

Chapter 18, “Overview of the Robot Application Functions”

Chapter 19, “Writing New Robot Application Functions”

Search Engine Robot Overview

This chapter describes the search engine robot. This chapter covers the following topics:

- [Introduction](#)
- [How the Robot Works](#)
- [Robot Configuration Files](#)
- [The Filtering Process](#)
- [Stages in the Filter Process](#)

Introduction

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.

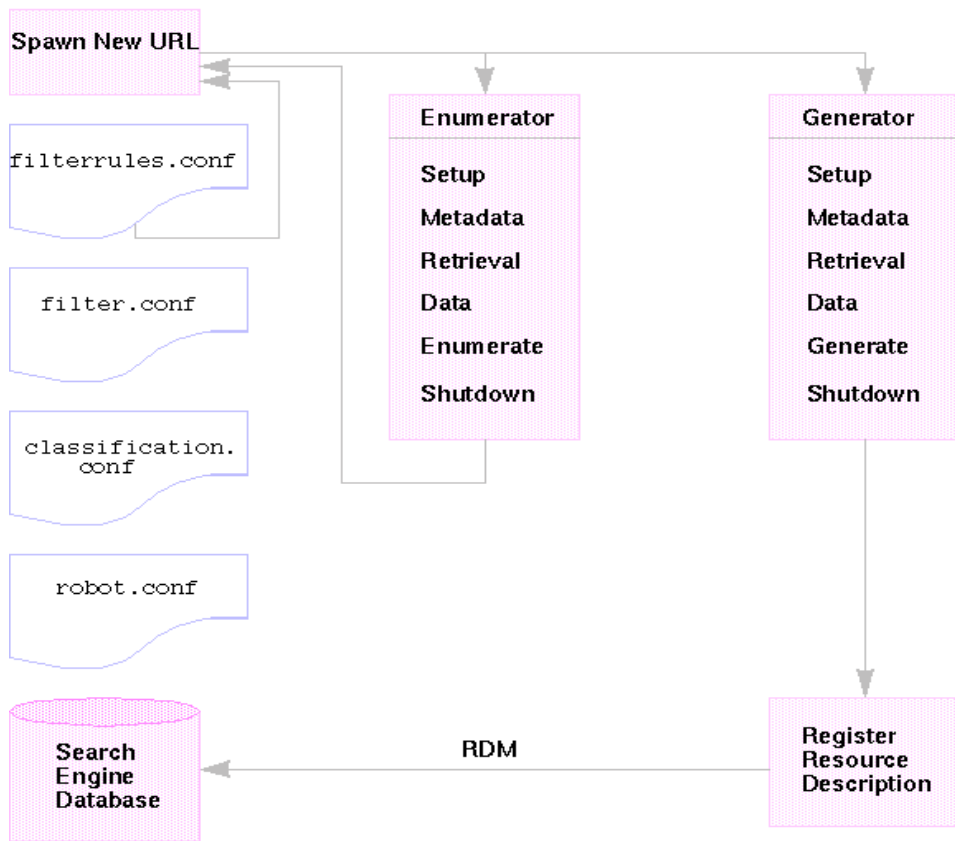
The enumerator filter locates resources by using network protocols. It tests each resource, and, if it meets the selection criteria, it is enumerated. For example, the enumerator filter can extract hypertext links from an HTML file and use the links to find additional resources.

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

Figure 16-1 illustrates how the search engine robot works. In Figure 16-1, 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 16-1 How the Robot Works



Robot Configuration Files

Robot configuration files define the behavior of the search engine robots. These files reside in the directory `webcontainer/portal/config`. The following table lists the configuration file in the first (left) column and provides a description of the corresponding robot configuration file in the second (right) column:

<code>robot.conf</code>	Defines most of the operating parameters for the robot.
<code>filter.conf</code>	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.
<code>filterrules.conf</code>	Contains the starting points (also referred to as starting point URLs) and rules used by the <code>filterrules-process</code> function.
<code>classification.conf</code>	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. [Table 16-1](#) lists examples of common metadata types:

Table 16-1 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.

Stages in the Filter Process

Robot Completion Scripts

This chapter contains the following sections:

- [Introduction](#)
- [Monitoring cmdHook Execution](#)
- [Preparing Your Completion Script to Appear in the Administration Interface](#)

Introduction

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 216](#).

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/portal`. This is important to keep in mind if you are using any relative directory references.

There are two examples provided in `portal-server-install-root/SUNWps/samples/robot`. They are a simple 'touch' test (`cmdHook0`) and an email upon completion script (`cmdHook1`).

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 would look like the following:

```
[12:45:57] Run cmd: /opt/SUNWps/bin/cmdHook1
[12:45:58] Complete cmd: /opt/SUNWps/bin/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 *portal-server-install-root*/SUNWps/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"
```

Preparing Your Completion Script to Appear in the Administration Interface

Overview of the Robot Application Functions

This chapter contains the following sections:

- [Introduction](#)
- [Robot Plug-in API Overview](#)
- [The Robot Application Function Header Files](#)

Introduction

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).

NOTE For more information on the search engine administration interface and on the predefined robot application functions (RAFs), see the *Portal Server Administration Guide*.

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.

The following steps are a brief overview of the process for creating 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 `portal-server-install-root/SUNWps/sdk/robot/include` directory contains all the header files you need to include when writing your plug-in functions.

The `portal-server-install-root/SUNWps/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. The following topics are described:

- [Header File Hierarchy](#)
- [Header File Contents](#)

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:

BaseDir/SUNWps/sdk/robot/ include	Contains header files that define general purpose data structures and function prototypes.
BaseDir/SUNWps/sdk/robot/ include/base	Contains the <code>systems.h</code> header files that handle low-level, platform independent functions such as memory, file, and network access.
BaseDir/SUNWps/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 following table describes the header files in the `include` directory in the first (left) column and a brief description of the corresponding directory in the second (right) column:

<code>csinfo.h</code>	Contains functions for object typing, specifically for mapping files to MIME types.
<code>csmem.h</code>	Contains memory-related definitions.
<code>robotapi.h</code>	Contains the type definitions for structures and the return-code definitions for robot API functions.
<code>filterrules.h</code>	Contains type definitions for structures needed by filter rules.

The following table describes the header files in the `base` directory in the first (left) column and a brief description of the corresponding directory in the second (right) column:

<code>systems.h</code>	Contains functions that handle system information.
------------------------	--

The following table describes the header files in the `libcs` directory in the first (left) column and a brief description of the corresponding directory in the second (right) column:

<code>adt.h</code>	Contains type definitions and function prototypes for utilities needed by the robot, such as linked lists, queues, and hash tables.
<code>cs.h</code>	Contains a library of common functions used by the Search Engine.
<code>csidcf.h</code>	Contains configuration definitions for Search Engine.
<code>getopt.h</code>	Contains routines to get options from the command line, for example, command line <code>prog -n arg1 -p arg2</code> , to get <code>arg1</code> and <code>arg2</code> .
<code>log.h</code>	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.

The Robot Application Function Header Files

Writing New Robot Application Functions

This chapter contains the following sections:

- [Introduction](#)
- [Compiling and Linking your Code](#)
- [Loading Your Shared Object](#)
- [Using your New Robot Application Functions](#)

Introduction

When you write robot application functions, you must 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 *Portal Server 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 `portal-server-install-root/SUNWps/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 232](#).

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 `portal-server-install-root/SUNWps/sdk/robot/include/libcs` directory).

The names of these functions are all prefixed by `libcs_`.

The following table contains the parameter manipulation functions in the first (left) column and a description of the corresponding function in the second (right) column. See the `portal-server-install-root/SUNWps/sdk/robot/include/libcs/pblock.h` header file for full function signatures with return type and arguments.

<code>libcs_param_create</code>	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 <code>pb_param</code> structure.
<code>libcs_param_free</code>	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 <code>libcs_pblock_remove</code> function.
<code>libcs_pblock_create</code>	Creates a new parameter block with a hash table of a chosen size. Returns the newly allocated parameter block
<code>libcs_pblock_free</code>	Frees a given parameter block and any entries inside it.
<code>libcs_pblock_find</code>	Finds the entry with the given name in a pblock and returns its value, otherwise returns NULL.
<code>libcs_pblock_findval</code>	Finds the entry with the given name in a pblock, and returns its value, otherwise returns NULL.
<code>libcs_pblock_remove</code>	Behaves like the <code>libcs_pblock_find</code> function, but in addition, it removes the entry from the pblock.

<code>libcs_pblock_nninsert</code> and <code>libcs_pblock_nvinsert</code>	These parameters create a new parameter with a given name and value and insert it into a given parameter block. The <code>libcs_pblock_nninsert</code> function requires that the value be an integer, but the <code>libcs_pblock_nvinsert</code> function accepts a string.
<code>libcs_pblock_pinsert</code>	Inserts a parameter into a parameter block.
<code>libcs_pblock_str2pblock</code>	Scans the given string for parameter pairs in the format <code>name=value</code> or <code>name="value"</code> , adds them to a <code>pblock</code> , and returns the number of parameters added.
<code>libcs_pblock_pblock2str</code>	Places all of the parameters in the given parameter block into the given string. Each parameter is of the form <code>name="value"</code> 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 SOIF syntax.

Objects in SOIF syntax have a schema name, an associated URL, and a set of attribute-value pairs.

In the [Code Example 19-1](#), the schema name is `@DOCUMENT`, the URL is: <http://developer.siroe.com/docs/manuals/htmlguid/index.htm>, and the SOIF contains attribute-value pairs for title, author, and description.

Code Example 19-1 SOIF 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 SOIF. It also has an `rd` field, whose value is the SOIF for the resource. Once you get the SOIF for the resource, you can use the functions for working with SOIF that are defined in *portal-server-install-root/SUNWps/sdk/rdm/include/soif.h* file to get more information about the resource. (The file *robotapi.h* includes *soif.h*.)

For example, the macro `SOIF_Findval(soif, attribute)` gets the value of the given attribute in the given SOIF. [Code Example 19-2](#) uses this macro to print the value of the `META` attribute if it exists for the resource being processed:

Code Example 19-2 SOIF_Findval Macro Example

```
int my_new_raf(libcs_pblock *pb, CSFilter *csf, CSResource *csr)
  char *metavalue;
  if (metavalue = (char *)SOIF_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 SOIF objects, see [Chapter 9, “Using the RDM API to Access the Search Engine and Database in C.”](#)

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 *portal-server-install-root/SUNWps/sdk/robot/include/robotoapi.h*. The following table describes the response status codes after the robot has completed processing in the first (left) column and includes a description of the corresponding status code in the second (right) column:

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 *portal-server-install-root/SUNWps/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
```

[Code Example 19-3](#) shows a sample function definition:

Code Example 19-3 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;

/* 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 SOIF_Findval(soif, attribute) is defined
 * in sdk/rdm/include/soif.h. It gets the value of the
 * given attribute from the given resource.
 * The rd in the CSResource is a soif that describes the resource
 */

if(!(s = (char *)SOIF_Findval(csr->rd, src)))

```

Code Example 19-3 Robot Application Function Example (*Continued*)

```

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);
}

/* 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 *portal-server-install-root/SUNWps/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 19-1](#), the compiled object files `t.o` and `u.o` are linked to form a shared object called `test.so`.

Table 19-1 Options for linking

System	Compile options
Solaris	<code>ld -G t.o u.o -o test.so</code>

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 `wordcount` 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
```

Using your New Robot Application Functions

Using the SOIF API to Work with SOIF Objects

Chapter 20, “Overview of the SOIF API”

Chapter 21, “SOIF API”

Overview of the SOIF API

This chapter contains the following sections:

- [What is SOIF?](#)
- [Using the SOIF API](#)
- [An Introductory Example](#)
- [Getting Search Server Database Contents as a SOIFStream](#)

Introduction

Resource descriptions in the search engine database are described in SOIF, and so are Resource Description Messages (RDMs) that processes can use to exchange resource descriptions across a network.

The SOIF API provides routines for creating and modifying SOIF objects in C.

The SOIF API is defined in the `soif.h` file in the following directory in your search engine installation directory:

portal-server-install-root/SUNWps/sdk/rdm/include

This chapter is restricted to discussing the use of C functions that come with the search engine SOIF 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 SOIF data should use the UTF-8 character set. Note that UTF-8 is fully backward compatible with 7-bit ASCII SOIF.

What is SOIF?

SOIF 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 SOIF format is a basic attribute-value format. SOIF files look like text but should be treated as binary data and edited with care. SOIF files contain tabs, and many editors will convert tabs to spaces and corrupt the file. You can use SOIF-manipulation functions to create and modify SOIF objects so you do not have to write and edit them manually.

The following sample SOIF 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 SOIF 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 SOIF API

The SOIF API is defined in the `soif.h` header file in directory `portal-server-install-root/SUNWps/sdk/rdm/include`.

The SOIF API defines structures and functions for working with SOIF objects. For example, the following code uses the functions `SOIF_Create()` and `SOIF_InsertStr()` to create a SOIF and add some attribute-value pairs to it:

```
SOIF mysoif=SOIF_Create("DOCUMENT", "http://varrius/doc.htm");
```

```
SOIF_InsertStr(mysoif, "title", "All About Style Sheets");
SOIF_InsertStr(mysoif, "author", "Robin Styles");
SOIF_InsertStr(mysoif, "description", "All you need to know about style
sheets");
```

These commands create a SOIF 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 SOIF object contains attribute-value pairs, which are each represented as `SOIFAVPair` objects. Using the SOIF 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 SOIF objects can be grouped together into SOIF streams, which are represented by `SOIFStream` objects. A `SOIFStream` object provides functionality for handling a stream of SOIF objects. For example, you can use the stream to filter attributes, and print the desired attributes for every SOIF in the stream.

Thus, the relevant data structures when using the SOIF API include:

- `SOIF` — a SOIF object.
- `SOIFAVPair` — an attribute-value pair.
- `SOIFStream` — a stream of SOIF objects.

An Introductory Example

You will find several examples of the use of the SOIF API in `portal-server-install-root/SUNWps/sdk/rdm/examples`.

This section discusses an example that is similar to (but not necessarily identical to) `example1.c`. It shows how to iterate through a SOIF stream and print the URL and number of attributes of each SOIF in the stream.

This example assumes that you have already created a file containing a SOIF stream which is available on `stdin`. For example, you could have created a SOIF 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 `SOIF_ParseInitFile()` to create a SOIF stream from the standard input.

Code Example 20-1 Simple SOIF Stream Parsing Example

```

/* Example 1 - Simple SOIF Stream Parsing */

#include <stdio.h>
#include <stdlib.h>
#include "soif.h"

int main(int argc, char *argv[])
{
    /* Define a SOIFStream and SOIF */
    SOIFStream *ss;
    SOIF *s;
    char *titleptr;

    /* Open a SOIF stream that gets its SOIF from stdin */
    ss = SOIF_ParseInitFile(stdin);
    /* SOIFStream_IsEOS() checks if this is the end of the stream */

    while (!SOIFStream_IsEOS(ss)) {
        if (!(s = SOIFStream_Parse(ss)))
            /* Exit the loop if the SOIF is invalid */
            break;

        /* Print the URL for each SOIF (will be "-" if there is no URL)*/
        printf("URL = %s\n", s->url);

        /* Print the title if it exists. */
        titleptr = SOIF_Findval(s, "title");
        printf("Title = %s\n", titleptr ? titleptr : "(none)")

        /* Print the number of attributes in the SOIF*/
        printf("# of Attributes = %d\n", SOIF_GetAttributeCount(s));

        /* release the memory used by the SOIF */
        SOIF_Free(s);
    }
    /* Close the SOIFStream and exit */
    SOIFStream_Finish(ss);
    exit(0);
}

```

Getting Search Server Database Contents as a SOIFStream

You can retrieve the entire contents of the search engine database as a SOIF 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 `WebContainer/portal`.

From the `WebContainer/portal` directory, run the following command:

```
/var/opt/SUNWps/bin/rdmgr -U
```

Be sure that the environment variable `LD_LIBRARY_PATH` to `portal-server-install-root/lib`.

This command prints the database contents as a SOIFStream. You can pipe the output to a program that uses SOIFStream routines to parse the SOIFs in the stream.

Getting Search Server Database Contents as a SOIFStream

SOIF API

This chapter contains the functions and objects defined in the `soif.h` header file. It contains the following sections:

- [Functions and Objects](#)
- [SOIF Structure](#)
- [Attribute-Value Pair Routines](#)
- [Multi-valued Attribute Routines](#)
- [Stream Routines for Parsing and Printing SOIFs](#)
- [Memory Buffer Management](#)

Functions and Objects

[Table 21-1 on page 245](#) provides an alphabetized version of the functions and objects for your reference.

Table 21-1 Alphabetized Functions and Objects Defined in the `soif.h` File

SOIF function or object	Category
<code>append</code>	Memory Buffer Management
<code>increase</code>	Memory Buffer Management
<code>reset</code>	Memory Buffer Management
<code>SOIF_Apply</code>	SOIF Structure
<code>SOIF_AttributeCompare</code>	Attribute-Value Pair Routines

Table 21-1 Alphabetized Functions and Objects Defined in the soif.h File *(Continued)*

SOIF function or object	Category
SOIF_AttributeCompareMV	Multi-valued Attribute Routines
SOIF_Contains	Multi-valued Attribute Routines
SOIF_Create	SOIF Structure
SOIF_DeleteMV	Multi-valued Attribute Routines
SOIF_Find	SOIF Structure
SOIF_Findval	SOIF Structure
SOIF_FindvalMV	Multi-valued Attribute Routines
SOIF_Free	SOIF Structure
SOIF_GetAttributeCount	SOIF Structure
SOIF_GetAttributeSize	SOIF Structure
SOIF_GetTotalSize	SOIF Structure
SOIF_GetValueCount	SOIF Structure
SOIF_GetValueSize	SOIF Structure
SOIF_Insert	Multi-valued Attribute Routines
SOIF_InsertAVP	SOIF Structure
SOIF_InsertMV	Multi-valued Attribute Routines
SOIF_InsertStr	Attribute-Value Pair Routines
SOIF_IsMVAttribute	Multi-valued Attribute Routines
SOIF_Merge	SOIF Structure
SOIF_MVAttributeParse	Multi-valued Attribute Routines
SOIF_ParseInitFile	Stream Routines for Parsing and Printing SOIFs
SOIF_ParseInitStr	Stream Routines for Parsing and Printing SOIFs
SOIF_PrintInitFile	Stream Routines for Parsing and Printing SOIFs
SOIF_PrintInitFn	Stream Routines for Parsing and Printing SOIFs
SOIF_PrintInitStr	Stream Routines for Parsing and Printing SOIFs
SOIF_Remove	SOIF Structure
SOIF_Rename	Attribute-Value Pair Routines
SOIF_Replace	Attribute-Value Pair Routines
SOIF_ReplaceMV	Attribute-Value Pair Routines
SOIF_ReplaceStr	Attribute-Value Pair Routines

Table 21-1 Alphabetized Functions and Objects Defined in the soif.h File *(Continued)*

SOIF function or object	Category
SOIF_SqueezeMV	Attribute-Value Pair Routines
SOIFAVPair_Create	Attribute-Value Pair Routines
SOIFAVPair_Free	Attribute-Value Pair Routines
SOIFAVPair_IsMV	Multi-valued Attribute Routines
SOIFAVPair_NthValid	Multi-valued Attribute Routines
SOIFAVPair_NthValue	Multi-valued Attribute Routines
SOIFAVPair_NthVsize	Multi-valued Attribute Routines
SOIFBuffer_Create	Memory Buffer Management
SOIFBuffer_Free	Memory Buffer Management
SOIFStream_Finish	Stream Routines for Parsing and Printing SOIFs
SOIFStream_GetAllowed	Stream Routines for Parsing and Printing SOIFs
SOIFStream_GetDenied	Stream Routines for Parsing and Printing SOIFs
SOIFStream_IsAllowed	Stream Routines for Parsing and Printing SOIFs
SOIFStream_IsEOS	Stream Routines for Parsing and Printing SOIFs
SOIFStream_IsParsing	Stream Routines for Parsing and Printing SOIFs
SOIFStream_IsPrinting	Stream Routines for Parsing and Printing SOIFs
SOIFStream_Parse	Stream Routines for Parsing and Printing SOIFs
SOIFStream_Print	Stream Routines for Parsing and Printing SOIFs
SOIFStream_SetAllowed	Stream Routines for Parsing and Printing SOIFs
SOIFStream_SetDenied	Stream Routines for Parsing and Printing SOIFs
SOIFStream_SetFinishFn	Stream Routines for Parsing and Printing SOIFs

SOIF Structure

A SOIF 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. SOIF supports text and binary data, and attributes can have multiple values.

An example SOIF is the following:

```

@DOCUMENT { http://www.siroe.com/
  title{17}:  Welcome to Siroe!
  author{13}: Dot Punchcard
}

```

A SOIF 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 SOIF object contains a collection of `SOIFAVPair` objects, which each contain an attribute and one or more values. To access attribute values in a SOIF, use `SOIF_find()` to retrieve the `AVPair` for the given attribute, or use `SOIF_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 SOIF objects by using the `SOIF_create()` function. You can also read SOIF objects from a SOIF stream.

SOIF_Create

```
NSAPI_PUBLIC SOIF *SOIF_Create(char *schema_name, char *url)
```

Creates a SOIF structure with the given schema name and URL.

SOIF_Free

```
NSAPI_PUBLIC void SOIF_Free(SOIF *)
```

Frees the given SOIF structure.

SOIF_GetTotalSize

```
NSAPI_PUBLIC int SOIF_GetTotalSize(SOIF *s)
```

Gets the estimated total size of the SOIF in bytes.

SOIF_GetAttributeCount

```
NSAPI_PUBLIC int SOIF_GetAttributeCount(SOIF *s)
```

Gets the number of attributes in the SOIF.

SOIF_GetAttributeSize

```
NSAPI_PUBLIC int SOIF_GetAttributeSize(SOIF *s)
```

Gets the size of the attributes only.

SOIF_GetValueSize

```
NSAPI_PUBLIC int SOIF_GetValueSize(SOIF *s)
```

Gets the size of the values only.

SOIF_GetValueCount

```
NSAPI_PUBLIC int SOIF_GetValueCount(SOIF *s)
```

Gets the number of values only.

SOIF_Merge

```
NSAPI_PUBLIC int SOIF_Merge(SOIF *dst, SOIF *src);
```

Use this function to merge two SOIF objects (perform a Union of their attribute-values). It returns non-zero on error; otherwise, returns zero and the 'dst' SOIF object contains all the attribute-value pairs from the 'src' SOIF 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.

SOIF_Find

```
#define SOIF_Find(soif, attribute-name)
```

retrieves the AVPair for the given attribute in the given soif. For example, the following statement gets the AVPair for the title attribute in the soif s:

```
SOIFAVpair avp=SOIF_Find(s, "title");
```

SOIF_Findval

```
#define SOIF_Findval(soif, attribute-name)
```

retrieves the value string for the given attribute in the given soif. For example, the following statement prints the value of the title attribute of the soif s:

```
printf("Title = %s\n", SOIF_Findval(s, "title"));
```

SOIF_Remove

```
#define SOIF_Remove(soif, attribute-name)
```

Removes the given attribute from the given soif.

SOIF_Insert

```
#define SOIF_Insert(soif, attribute-name, value, value-size)
```

Inserts the given attribute and the value of the given size as an AVPair into the soif.

SOIF_InsertAVP

```
#define SOIF_InsertAVP(soif, avpair)
```

Inserts the given AVPair into the given soif.

SOIF_Apply

```
#define SOIF_Apply(soif, function, user-data)
```

Applies the given function with the given argument (user-data) to each AVPair in the given soif. For example:

```
void print_av(SOIF *s, SOIFAVPair *avp, void *unused)
{printf("%s = %s\n", avp->attribute, avp->value);}

/* print every attribute and value in the soif s */
SOIF_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 SOIFAVPair structures.

The important fields in a SOIFAVPair structure are the following:

<code>char *attribute;</code>	Attribute string; '\0' terminated
<code>char *value;</code>	primary value; may be '\0' terminated
<code>size_t vsize;</code>	# of bytes (8 bits) for primary value
<code>char **values;</code>	Multiple values for multivalued attributes
<code>size_t *vsizes;</code>	the sizes for the values
<code>int nvalues;</code>	number of values associated with attribute
<code>int last_slot;</code>	last valid slot - array may contain holes

SOIFAVPair_Create

```
NSAPI_PUBLIC SOIFAVPair * SOIFAVPair_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.

SOIFAVPair_Free

```
NSAPI_PUBLIC void SOIFAVPair_Free(SOIFAVPair *avp);
```

Frees the memory used by the given SOIFAVPair structure

SOIF_Replace

```
NSAPI_PUBLIC int SOIF_Replace(SOIF *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 SOIF *s*.

SOIF_InsertStr

```
#define SOIF_InsertStr(soif, attribute, value)
```

Inserts the given attribute with the given value into the soif.

SOIF_ReplaceStr

```
#define SOIF_ReplaceStr(soif, attribute, value)
```

Replaces the existing value of the given attribute in the soif with the given value.

SOIF_Rename

```
NSAPI_PUBLIC int SOIF_Rename(SOIF *s, char *old_attr, char *new_attr);
```

Renames the given attribute to the given new name.

SOIF_AttributeCompare

```
NSAPI_PUBLIC int SOIF_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.

Table 21-2

AttributeA	AttributeB	Does SOIF_AttributeCompare() consider them to be the same?
title	Title	yes
title	Title	yes

Table 21-2

AttributeA	AttributeB	Does SOIF_AttributeCompare() consider them to be the same?
title	title	yes
title	title-page	no
title	title	no
author	title	no

Multi-valued Attribute Routines

A SOIF attribute can have multiple values. SOIF 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 SOIF 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
```

SOIF_AttributeCompareMV

```
NSAPI_PUBLIC int SOIF_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 `SOIF_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.

SOIF_MVAttributeParse

```
NSAPI_PUBLIC int SOIF_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.

SOIF_IsMVAttribute

```
NSAPI_PUBLIC char *SOIF_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.

SOIF_InsertMV

```
NSAPI_PUBLIC int SOIF_InsertMV(SOIF *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-multivalued form). If set, the *useval* flag tells the function to use the given value buffer rather than creating its own copy.

For example:

```
SOIF_InsertMV(s, "classification", 3, "web crawler", strlen("web crawler"));
```

inserts

```
classification-3{10}: web crawler
```

SOIF_ReplaceMV

```
NSAPI_PUBLIC int SOIF_ReplaceMV(SOIF *s, char *a, int slot, char *v, int
vsz, int useval);
```

SOIF_DeleteMV

```
NSAPI_PUBLIC int SOIF_DeleteMV(SOIF *s, char *a, int slot)
```

Deletes the value at the index slot in the attribute *a*. For example:

```
SOIF_DeleteMV(s, "classification", 3)
```

deletes classification-3.

SOIF_FindvalMV

```
NSAPI_PUBLIC const char *SOIF_FindvalMV(SOIF *s, const char *a, int slot)
```

Finds the value at the index slot in the attribute *a*. For example:

```
SOIF_FindvalMV(s, "classification", 3)
```

returns web crawler (using the previous example).

SOIF_SqueezeMV

```
NSAPI_PUBLIC void SOIF_SqueezeMV(SOIF *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 `SOIF_InsertMV()` invocations. For example, to insert values explicitly for the multivalued attribute `author-*`:

```
SOIF_InsertMV(s, "author", 1, "John", 4, 0);
SOIF_InsertMV(s, "author", 2, "Kevin", 5, 0);
SOIF_InsertMV(s, "author", 6, "Darren", 6, 0);
SOIF_InsertMV(s, "author", 9, "Tommy", 5, 0);
SOIF_FindvalMV(s, "author", 9); /* == "Tommy" */
SOIF_SqueezeMV(s);
SOIF_FindvalMV(s, "author", 9); /* == NULL */
SOIF_FindvalMV(s, "author", 4); /* == "Tommy" */
```

SOIFAVPair_IsMV

```
#define SOIFAVPair_IsMV(avp)
```

Use this to determine if the `AVPair` has multiple values or not.

SOIFAVPair_NthValid

```
#define SOIFAVPair_NthValid(avp,n)
```

Use this to determine if the `Nth` value is valid or not.

SOIFAVPair_NthValue

```
#define SOIFAVPair_NthValue(avp,n) ((avp)->values[n])
```

Use this to access the `Nth` value. For example:

```
for (i = 0; i <= avp->last_slot; i++)
    if (SOIFAVPair_NthValid(avp, i))
        printf("%s = %s\n", avp->attribute,
              SOIFAVPair_NthValue(avp, i));
```

SOIFAVPair_NthVsize

```
#define SOIFAVPair_NthVsize(avp,n) ((avp)->vsizes[n])
```

Use this to get the size of the `Nth` value.

SOIF_Contains

```
NSAPI_PUBLIC boolean_t SOIF_Contains(SOIF *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 SOIF `s`.

Stream Routines for Parsing and Printing SOIFs

A `SOIFStream` contains one or more SOIF objects.

The general approach is that you use SOIF streams to create and process streams of many SOIF objects. Given a SOIF stream, you can parse it to get the SOIF objects from it. Use the `parse()` routine to get the next SOIF object in a SOIF stream. You can use `SOIFStream_IsEOS()` to check whether the last object has been parsed.

You can use filtering functions for a SOIF stream to specify that certain SOIF attributes are allowed or denied. If an attribute is allowed, you can parse and print that attribute for SOIF objects in the stream. If it is denied, you cannot parse or print that attribute of SOIF objects in the stream.

SOIF streams can be disk or memory based.

When you create a `SOIFStream`, you need to specify if you will be printing or parsing the SOIF 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 SOIF stream.

For creating a SOIF streams into which you will be printing SOIFs, the functions are the following:

<code>SOIF_PrintInitFile()</code>	creates a disk-based stream ready for printing.
<code>SOIF_PrintInitStr()</code>	creates a memory-based stream ready for printing.
<code>SOIF_PrintInitFn()</code>	creates a generic application-defined stream ready for printing. The given <code>'write_fn'</code> is used to print the stream.

To create SOIF stream from a file or a string containing SOIF, use the following functions:

<code>SOIF_ParseInitFile()</code>	creates a disk-based stream ready for parsing. The stream is created from an input containing SOIF syntax.
-----------------------------------	--

`SOIF_ParseInitStr()` creates a memory-based stream ready for parsing. The stream is created from an input containing SOIF syntax.

SOIFStream objects have a caller-data field, which you can use as you like:

```
void *caller_data; /* hook to be used by caller */
```

Use `SOIFStream_Parse()` to get the SOIF objects from the SOIF stream, and use `SOIFStream_Print()` to write SOIF objects to the SOIF stream.

When you've finished with the stream, close it by using `SOIFStream_Finish()`. Use `SOIFStream_SetFinishFn()` to trigger the given `finish_fn` function.

The following example code takes a SOIF stream in `stdin` and prints each SOIF in the stream to `stdout`. Notice that this code uses `SOIF_ParseInitFile()` to create the SOIFStream to parse the input file, and uses `SOIF_PrintInitFile()` to create the stream to print the SOIFs to `stdout`.

```
SOIFStream *soifin = SOIF_ParseInitFile(stdin);
SOIFStream *soifout = SOIF_PrintInitFile(stdout);
SOIF *s;
while (!SOIFStream_IsEOS(soifin)) {
    if ((s = SOIFStream_Parse(soifin)) {
        SOIFStream_print(soifout, s);
        SOIF_Free(s);
    }
}
```

SOIF_PrintInitFile

```
NSAPI_PUBLIC SOIFStream *SOIF_PrintInitFile(FILE *file)
```

Creates a disk-based stream ready for printing.

SOIF_PrintInitStr

```
NSAPI_PUBLIC SOIFStream *SOIF_PrintInitStr(SOIFBuffer *memory)
```

Creates a memory-based stream ready for printing.

SOIF_PrintInitFn

```
NSAPI_PUBLIC SOIFStream *SOIF_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.

SOIF_ParseInitFile

```
NSAPI_PUBLIC SOIFStream *SOIF_ParseInitFile(FILE *fp)
```

Creates a disk-based stream ready for parsing. The file must contain SOIF-formatted data. The function reads SOIF data from the file object `fp`.

SOIF_ParseInitStr

```
NSAPI_PUBLIC SOIFStream *SOIF_ParseInitStr(char *buf, int bufsz)
```

Creates a memory-based stream ready for parsing. The character buffer must contain SOIF-formatted data.

SOIFStream_Finish

```
NSAPI_PUBLIC int SOIFStream_Finish(SOIFStream *)
```

Closes the stream when you have finished with it.

SOIFStream_SetFinishFn

```
NSAPI_PUBLIC int SOIFStream_SetFinishFn(SOIFStream *, int
(*finish_fn)(SOIFStream *))
```

Allows you to hook up a function for cleaning up after the SOIF stream finishes its business. The `finish_fn` will be called when `SOIFStream_Finish()` has finished executing.

SOIFStream_Print

```
#define SOIFStream_Print(ss, s)
```

Prints another SOIF object to the SOIF stream `ss`. Returns 0 on success, or non-zero on error.

SOIFStream_Parse

```
#define SOIFStream_Parse(ss)
```

Parses and returns the next SOIF object in the SOIF stream.

SOIFStream_IsEOS

```
#define SOIFStream_IsEOS(s)
```

Returns 1 (true) if the SOIF stream has been exhausted.

SOIFStream_IsPrinting

```
#define SOIFStream_IsPrinting(s)
```

Returns 1 (true) if the SOIF has been set up in a stream by `SOIF_PrintInitFile()` or `SOIF_PrintInitStr()`.

SOIFStream_IsParsing

```
#define SOIFStream_IsParsing(s)
```

Returns 1 (true) if the SOIF has been setup in a stream by `SOIF_ParseInitFile()` or `SOIF_ParseInitStr()`.

Filtering SOIF Objects

To support targeted parsing and printing, you can use the attribute filtering mechanisms in the SOIF stream. For each SOIF stream object, you can associate a list of allowed attributes. When printing a SOIF stream, only the attributes that match the allowed attributes will be printed. When parsing a SOIF stream, only the attributes that match the allowed attributes will be parsed.

`SOIFStream_IsAllowed()` and `SOIFStream_SetAllowed()` allow attributes, while `SOIFStream_IsDenied()` and `SOIFStream_SetDenied()` deny attributes. You can allow or deny an attribute, but not both.

SOIFStream_IsAllowed

```
NSAPI_PUBLIC boolean_t SOIFStream_IsAllowed(SOIFStream *ss, char
*attribute);
```

Indicates that the given attribute is allowed (that is, it can be printed or parsed).

SOIFStream_SetAllowed

```
NSAPI_PUBLIC int SOIFStream_SetAllowed(SOIFStream *ss, char
*allowed_attrs[])
```

Sets all the attributes in the `allowed_attrs` array to allowed.

SOIFStream_SetDenied

```
NSAPI_PUBLIC int SOIFStream_SetDenied(SOIFStream *ss, char
*denied_attrs[]);
```

Sets all the attributes in the `allowed_attrs` array to be denied (that is, they cannot be parsed or printed).

SOIFStream_GetAllowed

```
NSAPI_PUBLIC char **SOIFStream_GetAllowed(SOIFStream *ss)
```

Returns an array of all the attributes that are allowed.

SOIFStream_GetDenied

```
NSAPI_PUBLIC char **SOIFStream_GetDenied(SOIFStream *ss);
```

Returns an array of all the attributes that are denied.

Memory Buffer Management

You can use SOIF 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 SOIF routines to use.

A SOIF Buffer is represented in a `SOIFBuffer` structure, that is created with the `SOIFBuffer_Create()` function and freed with the `SOIFBuffer_Free()` function. The `SOIFBuffer` structure provides the `append()`, `increase()`, and `reset()` functions for manipulating the data in the buffer.

SOIFBuffer_Create

```
NSAPI_PUBLIC SOIFBuffer *SOIFBuffer_Create(int default_sz);
```

The `SOIFBuffer` is used in `SOIF_PrintInitStr(SOIFBuffer *memory)`. Before you can print SOIF to memory, you need to create a buffer for output.

SOIFBuffer_Free

```
NSAPI_PUBLIC void SOIFBuffer_Free(SOIFBuffer *sb);
```

Releases the memory buffer created by `SOIFBuffer_Create()`.

append

```
void (*append)(SOIFBuffer *sb, char *data, int n)
```

Copies `n` bytes of data into the buffer.

increase

```
void (*increase)(SOIFBuffer *sb, int add_n)
```

Increases the size of the data buffer by `add_n` bytes.

reset

```
void (*reset)(SOIFBuffer *sb)
```

Resets the size of the data buffer and invalidates all currently valid data. A buffer can be reused by resetting it this way.

Using the RDM API to Access the Search Engine and Database in C

Chapter 22, “What is an RDM?”

Chapter 23, “About the RDM API”

Chapter 24, “Example of Submitting a Query”

What is an RDM?

This chapter contains the following sections:

- [Introduction](#)
- [RDM Format Syntax](#)

Introduction

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 SOIF.

RDM Header

The RDM header section begins with a SOIF object whose schema name is `RDMHEADER` which must contain at least the attributes listed in the following table:

<code>rdm-version</code>	A string identifying the version of the message specification. (for example, 1.0).
<code>rdm-type</code>	A string identifying the nature of this message.
<code>rdm-query-language</code> (required only for Request messages)	A string which identifies which query language is used in the given request.
<code>catalog-service-id</code> (optional)	A CSID identifying the catalog to which the request/response applies (for example, <code>x-catalog://siroe.budgie.com:80/finance</code> . 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 SOIF 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.

About the RDM API

This chapter

- [Introduction](#)
- [rdm.h File](#)
- [API Reference](#)

Introduction

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 SOIF 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:

portal-server-install-root/SUNWps/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 SOIF:

```
@RDMQUERY { -
    scope{7}:varrius
}
```

The following statement:

```
RDMQuery_SetViewAttr(myquery, "URL,Title");
```

modifies the `RDMQuery` so that it corresponds to the following SOIF:

```
@RDMQUERY { -
    scope{7}:varrius
    view-attributes{30}: URL,Title,Author,Last-Modified
}
```

Both the `RDMHeader` and `RDMQuery` structures have `soif` fields, which contain the SOIF data for the header or the query. To extract SOIF data from `RDMHeader` or `RDMQuery`, you can access the `soif` field. Thus, you can also use the RDM SOIF API to create and modify `RDMHeader` and `RDMQuery` objects.

API Reference

This section describes the RDM API, located in directory `portal-server-install-root/SUNWps/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 SOIF arguments, and an `RDMX_Create()` function, which takes non-SOIF 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 SOIF object into an existing RDMX structure.

RDMHeader

An `RDMHeader` represents the header of an RDM. An `RDMHeader` structure has one public field, `soif`, which is a SOIF containing a collection of attribute-value pairs. The allowable attribute names of the attribute-value pairs in the SOIF 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 following table describes the allowable values for `RDM-Type` in the first (left) column and includes a description of the corresponding `RDM-Type` value in the second (right) column:

<code>RDM_MSGTYPE_UNKNOWN</code>	Undefined or unknown message type
<code>RDM_RD_UPD_REQ</code>	Requesting an RD update
<code>RDM_RD_REQ</code>	Requesting an RD update (same as <code>RDM_RD_UPD_REQ</code>)
<code>RDM_RD_DEL_REQ</code>	Requesting an RD delete
<code>RDM_RD_UPD_RES</code>	Responding to an RD update request
<code>RDM_RD_RES</code>	Responding to an RD update request (same as <code>RDM_RD_UPD_RES</code>)
<code>RDM_RD_DEL_RES</code>	Responding to an RD delete request
<code>RDM_SD_REQ</code>	Requesting a server description
<code>RDM_SD_RES</code>	Responding to a server description request
<code>RDM_SCH_REQ</code>	Requesting a schema description
<code>RDM_SCH_RES</code>	Responding to a schema description request
<code>RDM_TAX_REQ</code>	Requesting a taxonomy description
<code>RDM_TAX_RES</code>	Responding to a taxonomy description request
<code>RDM_STAT_REQ</code>	Requesting a status
<code>RDM_STAT_RES</code>	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 RDMDType */
RDMHeader_GetVersion(RDMHeader)
RDMHeader_GetQueryLanguage(RDMHeader)
RDMHeader_GetCSID(RDMHeader)
RDMHeader_GetResponseInterpret(RDMHeader)
RDMHeader_GetErrorMessage(RDMHeader)
RDMHeader_GetErrorNumber(RDMHeader)

RDMHeader_SetType(RDMHeader, RDMDType)
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(SOIFStream *ss);
```

RDMHeader_Create

```
NSAPI_PUBLIC RDMHeader *RDMHeader_Create(RDMDType t);
```

RDMHeader_CreateRequest

```
NSAPI_PUBLIC RDMHeader *RDMHeader_CreateRequest(RDMDType, char *ql, CSID *)
```

RDMHeader_CreateResponse

```
NSAPI_PUBLIC RDMHeader *RDMHeader_CreateResponse(RDMDType, char *ri, CSID
*);
```

RDMHeader_Free

```
NSAPI_PUBLIC int RDMHeader_Free(RDMHeader *r);
```

RDMHeader_Merge

```
NSAPI_PUBLIC int RDMHeader_Merge(RDMHeader *, SOIF *)
```

Merges data from a SOIF object into the RDMHeader object.

RDMQuery

An `RDMQuery` represents the body of an RDM. An `RDMQuery` structure has one public field, `soif`, which is a SOIF containing a collection of attribute-value pairs. The allowable attribute names of the attribute-value pairs in the SOIF 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(SOIFStream *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 *, SOIF *);
```

Other RDM Structures

In addition to `RDMHeader` and `RDMQuery`, the file `rdm.h` provides definitions and functions for the following auxiliary objects. See the file `rdm.h` in *portal-server-install-root/SUNWps/sdk/rdm/include* for details.

- `RDMRequest`
- `RDMResponse`
- `RDMServer`
- `RDMView`
- `RDMViewHits`
- `RDMViewOrder`
- `RDMTaxonomy`
- `RDMClassification`
- `RDMSchema`

Example of Submitting a Query

This chapter contains the following sections:

- [Introduction](#)
- [RDMHeader and RDMQuery Object Parameters](#)
- [Running the Example](#)

Introduction

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 `portal-server-install-root/SUNWps/lib` directory). This program takes an RDM request (which is a message in SOIF format), sends it to the search engine, and outputs the results as a SOIF stream. You can also use HTTP to post an RDM stream directly to the server through its URL (`http://server:port/portal/search`). The program must be run in a search-enabled Sun Java System Portal Server software instance directory such as the default `/var/opt/SUNWps/serverinstance/portal` 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 following table describes the `RDMHeader` and `RDMQuery` object parameters in the first (left) column and provides a description of the corresponding parameter in the second (right) column

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 <code>/var/opt/SUNWps/<i>serverinstance</i>/portal/config/search.conf</code> . For SSL enabled server instances, use <code>x-catalogs</code> instead of <code>x-catalog</code> in the CSID.
MY_SCOPE	Query string. The default is to search for documents containing the string <code>varrius</code> . For example, if you want to search for all documents containing the string <code>style_sheets</code> , then set <code>MY_SCOPE</code> to <code>style_sheets</code> .
MY_ATTR_VIEW	Attributes to be printed, such as URL, title and description.

Code Example 24-1 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.

Code Example 24-1 RDM API to Submit a Query Example

```

/***** Example Use of the RDM API to submit a query *****/

#include <stdio.h>
#include "soif.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;
    SOIFStream *out = SOIF_PrintInitFile(stdout);

    /* Create the RDMQuery and specify its scope */
    if(!(myquery = RDMQuery_Create(MY_SCOPE))) {
        perror("RDMQuery_Create\n");
        exit(-1);
    }
}

```

Code Example 24-1 RDM API to Submit a Query Example (*Continued*)

```

/* 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))) {
    perror("RDMHeader_CreateRequest\n");
    exit(-1);
}

/* print the RDMHeader to the output SOIF stream */
/* print the RDMQuery to the output SOIF stream */
(*out->print)(out, myheader->soif);
(*out->print)(out, myquery->soif);

/* free the structures and exit */
RDMHeader_Free(myheader);

RDMQuery_Free(myquery);
SOIFStream_Finish(out);
exit(0);
}
/***** EOF *****/

```

Running the Example

To run the example described in [Code Example 24-1 on page 276](#), follow these steps:

1. Edit the definitions for `MY_SCOPE`, `MY_CSID`, and `MY_ATTR_VIEW` as appropriate for your situation. For more information, see [Code Example 24-2](#).
2. Save the file in `portal-server-install-root/SUNWps/sdk/rdm/examples` directory. For example, save the file as `example4.c` in `portal-server-install-root/SUNWps/sdk/rdm/examples` directory.
3. Create a makefile. You can find sample makefiles at `portal-server-install-root/SUNWps/sdk/rdm/examples` directory. Edit the makefile to include `example4.c`. [Code Example 24-2](#) shows the makefile with the changes needed for `example4.c` in bold:

Code Example 24-2 Makefile for SOIF/RDM examples

```

# Makefile for SOIF/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 *portal-server-install-root/SUNWps/sdk/rdm/examples* directory, build the example as follows:

```
Solaris:    gmake
```

5. From the *portal-server-install-root/SUNWps/sdk/rdm/examples* directory, run the `example4.c` program to generate the RDM file.

```
example4.c > rdm.soif
```

The file `rdm.soif` will look like the following example:

```

@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.soif` created in must be placed into the server instance directory.

6. Send the SOIF contained in `rdm.soif` to the program `sendrdm`. For example, type:

```
./run-cs-cli sendrdm -u /portal/search rdm.soif
```

The current directory should be the server instance directory. If `rdm.soif` is not in the server instance directory, reference the file from where it was created.

For example:

- a. Change directories to `server_instace_dir`.
- b. Type `./run-cs-cli sendrdm -u /portal/search sdk_dir/rdm.soif`.
- c. The results of the `sendrdm` program will be a SOIF 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 `SOIFStream`) to another program to print the results of the query.

Using Java To Access the Search Engine Database

Chapter 25, “Overview of the Search Engine Java SDK”

Chapter 26, “Using Java To Access the Search Server Database”

Chapter 27, “Using Java To Add Entries to the Search Engine Database”

Overview of the Search Engine Java SDK

This chapter contains the following sections:

- [Introduction](#)
- [The Search Engine Java SDK](#)
- [Running the Sample Applications](#)

Introduction

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:

portal-server-install-root/SUNWps/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 SOIF format, where SOIF stands for Summary Object Interchange Format.

For a discussion of SOIF, see:

<http://www.w3.org/TR/NOTE-rdm.html#soif>.

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 appletviewer 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 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 .:searchsdk.jar com.sun.portal.search.demo.SearchDemo  
http://portal_server_host_name:port/portal/search 'search query'
```

The codebase parameter indicates the directory containing the main class and the ancillary classes for the applet. You can specify this in the form:

```
"http://host_name:compass_server_port/java"
```

► **To Install and Run the Search Demo Applet**

To run the search demo as an applet, use the JDK `appletviewer` or a browser with the Java 1.3.1 plugin. You must also edit `SearchDemo.html` to configure the information about your search engine location.

NOTE The instructions included here are for `appletviewer`. To run the applet in a browser, you need Java 1.2 or later browser plugin. More information on the Java browser plugin 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 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:

1. 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.

2. 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
```


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](#)
- [Executing A Query and Getting the Results](#)
- [Working Through An Example Search Application](#)

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 are outlined in the following table:

<code>String scope</code>	The query string or scope, that is, the string being searched for.
<code>String viewAttributes</code>	A comma-delimited list of the SOIF attributes to be retrieved from the database, such as URL, Author, Description. For example, <code>score,url,title,description,classification</code> .
<code>String viewOrder</code>	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, <code>-score,+title</code> .
<code>int firstHits</code>	The hit number of the first result to return. A typical value is 1.
<code>int viewHits</code>	Maximum number of results to return. A typical value is 10.
<code>String queryLanguage</code>	The Search Server query language. You should use search for a normal query.
<code>String database</code>	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.
<code>String RDMServer</code>	The URL of the search engine servlet. This argument has the form: <code>http://hostname.domain.com:port/portal/search</code> For example, if the search server is installed on <code>www.yourcompany.com</code> on port 80, the value would be: <code>http://www.yourcompany.com:80/portal/search</code>
<code>String ssoToken</code>	An Sun Java System Identity Server software single sign on token used when doing secure searches. There is also a simpler convenience constructor with the following syntax: <code>public Search(String scope, String RDMServer)</code>

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 SOIF 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 SOIFInputStream getResultStream()
```

The function `getResultStream()` returns a `SOIFInputStream` which is used to read the SOIF hit objects. Each SOIF 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 `SOIFInputStream` 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 `soif` 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.soif.*;

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.

Code Example 26-1 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.
 */
```

Code Example 26-1 SearchDemo (Continued)

```

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 SOIFOutputFile = null;
        if (args != 1 && args != 2 && args != 3) {
            System.out.println("args: RDMServer [query]
            [soif_output_file_name]");
            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.setSOIFfile(argv[2]); // dump raw soif 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.setSOIFfile(String filename) method is used by the main program to direct search results to a file when running in command line mode.

Code Example 26-2 SimpleSearch Class

```

/** Performs a simple search and displays its results. */
class SimpleSearch {
    String RDMServer;
    String SOIFOutputFile;
    /**
     * 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 SOIF results into - only
     * use if running from the comand line or an applet with file
     * system access
     */
    public void setSOIFfile(String filename) {
        SOIFOutputFile = filename;
    }
    /** Execute a search */
    public void doSearch(String scope) throws IOException {
        ...see Code Example 26-3...
    }
}

```

Before submitting the search, SimpleSearch needs to create a Search object. The constructor for the Search class takes several arguments as discussed previously.

Code Example 26-3 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
    ** 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.

<code>viewOrder = "-score"</code>	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.
<code>firstHit = 1</code>	The hit number of the first returned result.
<code>viewHits = 20</code>	The maximum number of results to return.
<code>queryLanguage = "search"</code>	The search server query language. Use search for normal searches.
<code>RDMServer</code>	The URL of the remote search engine, specified as an argument to the SimpleSearch constructor.
<code>ssoToken = null</code>	The Identity Server 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 (SOIFOutputFile != null) {
    try {
        sos = new DataOutputStream(new FileOutputStream(SOIFOutputFile));
    }
    catch (Exception e1) {
        System.out.println("Error: failed to create output file: " + e1);
    }
}

int pagenum = 1;
int pagesize = 10;
SOIFBuffer firstPageSOIF = new SOIFBuffer();
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 `SOIFInputStream` object. Each result is read from this `SOIF` 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 `SOIF` 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 `SOIF` 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 SOIF objects and print its URL, title, description, and score to the output stream (either the Java console, standard output, or a named output file).

```

SOIFInputStream resultStream = search.getResultStream();
SOIF soif;
/* Examine the results of the search. The following
 * code loops through the stream of SOIF instances. */
for (soif = resultStream.readSOIF(); soif != null; soif =
resultStream.readSOIF()) {
    // For illustration, dump out the entire SOIF on the first page only.
    if (pagenum == 1)
        firstPageSOIF.write(soif.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 = soif.getURL();
    String t  = soif.getValue("title");
    String d  = soif.getValue("description");
    String sc = soif.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 SOIF output file, write the SOIF data there too...
    if (sos != null) {
        try {
            sos.writeBytes(soif.toString());
        }
        catch (Exception e1) {
            System.out.println("Error: failed to write to SOIF
output file: " + e1);
        }
    }
}
// Break if the largest requested hit has been displayed
if (search.getHitCount() <= (firstHit + pagesize - 1))
    break;
}
if (firstPageSOIF == null)
    System.out.println("No matching documents found.");
}

```

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](#)
- [SOIF Object](#)
- [Constructing and Submitting a Request](#)

rdmgr

The `rdmgr` utility can add new data as well as replace, modify, or retrieve existing data. All data input and output is done using SOIF, with UTF-8 character encoding for character fields. Note that SOIF also supports binary-valued fields and they can be added or retrieved too.

For more information on `rdmgr`, see *Portal Server Administration Guide*.

In the simplest case, `rdmgr` can be used to add a file containing multiple SOIF Resource Descriptions (RDs) to the database. This is as simple as creating a SOIF file with the search sdk, or by other means, and adding the data with the command `rdmgr soif_input_file`. The `rdmgr` also accepts resource description submit requests as input. Submit requests also use SOIF format and include a request header in addition to the normal body consisting of the SOIF data to be added or retrieved to or from the database.

SOIF Object

A SOIF 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.soif` package in the Search Server Java SDK is used to build SOIF objects and write them to a file.

Constructing and Submitting a Request

You can use the SOIF 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.

```
SOIF req = new SOIF("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."

```
SOIF data = new SOIF("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`:

```
SOIFOutputStream sos = new SOIFOutputStream("soif_file");
sos.write(req);
sos.write(data);
sos.close();
```

At this point `soif_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:

Code Example 27-1 `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:

<code>retrieve</code>	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.
<code>insert</code>	The server adds or replaces the RDs supplied to the database.
<code>delete</code>	The server deletes the RDs. As with <code>retrieve</code> , 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.

Localizing Sun Java System Portal Server

Chapter 28, “Templates and JSPs”

Chapter 29, “Properties”

Chapter 30, “Localization Support in PAPI”

Templates and JSPs

This chapter contains the following sections:

- [Desktop Templates](#)
- [File Lookup Mechanism](#)

This chapter addresses the Desktop templates and JavaServer Pages™ (JSP™) in the Portal Server software that can be translated to support localization.

Desktop Templates

Template Files

The `/etc/opt/SUNWps/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, `/etc/opt/SUNWps/desktop/default` is translated to `/etc/opt/SUNWps/desktop/default_locale`.

NOTE The *desktoptype* refers to the Desktop type attribute which is now a comma separated list. The usage of this attribute is described in Chapter 1 of the *Portal Server Desktop Customization Guide*.

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 hexadecimal 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 Java™ Development Kit (JDK) 1.3.1 package that is shipped with the Portal Server software.

JSPs

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 must look like 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.

Images

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 *portal-server-install-root/SUNWps/web-src* directory and then deploy it with the *portal-server-install-root/SUNWps/bin/deploy* `redeploy` 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 Identity Server) and searches for the file. For more information on the search order, see [“Presentation Method” on page 53](#).

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).

Properties

This chapter contains the following sections:

- [Resource Bundles](#)
- [Display Profile](#)

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 `/etc/opt/SUNWps/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 hexadecimal 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

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.

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

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, the following sample display profile fragment (stored in file `dp-locale.xml`) is needed:

```
<?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. To upload the XML display profile fragment using `dpadmin` (using the `add` subcommand with the `parent` option), type:

```
portal-server-install-root/SUNWps/bin/dpadmin add -p parent -u dn_amadmin -w password
-d distinguishedname dp-locale.xml
```

Localization Support in PAPI

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**

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**

See the Javadocs for more information on these methods.

Glossary

For a list of terms, please see <http://docs.sun.com/doc/816-6873>.

Index

A

Actions,DesktopServlet
content 47–48
edit 49–55
introduction 46
logout 60
process 56–59

Administration console 20

APIs

authentication 26
Desktop 21
Provider 22, 31
search service 24

AsciiFormInputExpectedException 38

Assumptions in this guide 9

Authentication 18, 26

B

Bold monospaced font usage in this guide 12

Building block providers 23

C

Channels

container (see Container channels)
introduction 16
leaf (see Leaf channels)

Client type methods,ProviderContext 35

ClientTypeFilters,ProviderContext 35

cmdHook 215, 216

Command-line prompts in this guide 12

Compiling a portlet 185, 193

Container channels
introduction 16

ContainerProvider interface 66, 67

ContainerProviderAdapter class 67, 68

ContainerProviderContext interface 67

D

Database,search service 25

Debug utility 20

DefaultRegistrationValidator class 202

Deploying a portlet 187, 195

Desktop

APIs 21

exceptions 37

introduction 16, 20

templates 305

user session 39

DesktopServlet

actions (see Actions,DesktopServlet)

introduction 23

overview 45

Display profile

channel properties 79

document 18

- document merging [18](#)
- introduction [17](#)
- mandatory properties [78](#)
- provider properties [77](#)
- xml fragments [82](#)

`dpadmin` command [84](#)

E

- EDIT_COMPLETE [36](#)
- EDIT_SUBSET [36](#)
- enumerator filter [209](#), [212](#)

F

- File lookup [75–77](#)
- File system,Portal Server software [27](#)

G

- generator filter [209](#), [212](#)
- GenericPortlet class [183](#), [190](#)

H

How-to

- create a new Search object [287](#)
- Install and run the Search demo command line program [284](#)
- run the search demo as an applet [285](#)
- send the RDMHeader and RDMQuery objects to the search engine to perform a query request [275](#)
- submit a query [289](#)

I

- Image files,Desktop [307](#)
- Interfaces overview,PAPI [33](#)
- InvalidEditFormDataException [38](#)
- Italicized font usage in this guide** [12](#)

J

- Java Developers Kit (JDK) [19](#)
- Java SDK,search engine [283](#), [287](#)
- Javadocs URL [15](#)
- JSP files,Desktop [306](#)
- JSPContainerProviderAdapter class [68](#)
- JSPProvider
 - extending [117](#)
 - introduction [62](#)
- JSPSingleContainerProvider class [66](#), [69](#), [133](#)
- JSPTabContainerProvider class [66](#), [71](#), [161](#)
- JSPTableContainerProvider class [66](#), [69](#), [151](#)

L

- Leaf provider/channels
 - introduction [16](#), [61](#)
- Leaf providers
 - extending [117](#)
- Logging [18](#)

M

- ModifiableTab [71](#)
- Monospaced font usage in this guide** [11](#)

O

Overview

- Portal Server software 15

P

PAPI

- base class relationship with leaf providers 61
- introduction 16
- overview 22, 31
- relationship between the various interfaces, classes, and exceptions 31

Policy management 20

Portal Server software

- architecture 19
- Desktop service 16
- file system 27
- filters 37
- overview 15
- purpose 16
- search service 18
- web container 27

Portlet

- compiling 185, 193
- converting providers to portlets 191
- deploying 187, 195
- mapping GenericPortlet class to ProviderAdapter class 190
- overview 181
- web application 185, 193

Profile management 18

ProfileProviderAdapter class 34, 96, 313

Properties

- display profile 77
- resource bundle 79

PropertiesFilter class 37, 113

Provider

- class loader 80
- custom class file location 80
- developing a custom provider 73
- required display profile properties 78

Provider Application Programming Interface (see PAPI)

Provider interface 33, 93

ProviderAdapter 34

ProviderAdapter class 33, 93, 96, 190

ProviderContext interface 34, 67, 313

ProviderContextException 34

ProviderEditTypes 36

ProviderException 34

ProviderException class 37, 38

ProviderWidths 36

R

RDM

- API 267

RDM API 269

rdmgr 243

- introduction 297

- types of requests 300

RDMS 239

Registration, WSRP

- in-band 200

- port type 200

- validation process 200

RegistrationValidator interface 200

Related information 12

Resource bundle

- file naming convention 310

- installation location 310

- overview 309

- properties 79, 310

Resource Description Manager (see RDM)

Resource Description Messages

- see RDMS 239

Resource Description Messages (RDMS) 267

Robot, search engine 25

RouterContainerProvider class 169

S

Search engine

- APIs [24](#)
- database [25](#)
- introduction [18, 20](#)
- Java SDK [283, 287](#)
- robot [25](#)
- sendrdm [280, 287](#)
- Service methods,ProviderContext [35](#)
- Servlet methods,ProviderContext [35](#)
- Session management [18](#)
- Session methods,ProviderContext [35](#)
- Session,user [39](#)
- Single-sign-on [18](#)
- SOIF [240](#)
- SOIF object [298](#)

T

- TabContainer interface [66, 70](#)
- Template files,Desktop [305](#)

U

- UnknownEditTypeException [38](#)
- UnmodifiableTab [70](#)
- URL for Javadocs [15](#)
- URL for this guide** [13](#)
- URLScrapperProvider
 - extending [124](#)
 - introduction [63](#)

W

- WAR file [307](#)
- Web container,Portal Server software [27](#)
- WSRP
 - in-band registration [200](#)
 - port type registration [200](#)
 - registration validation [200](#)

X

- XMLProvider
 - extending [128](#)
 - introduction [64](#)