

Oracle Portal-to-Go

Implementation Guide

Release 1.0.2.2

Part No. A86635-02

ORACLE[®]

Copyright © 2000 Oracle Corporation. All rights reserved.

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle Corporation disclaims liability for any damages caused by such use of the Programs.

Oracle is a registered trademark, and JDeveloper, Oracle7, Oracle8, Oracle8i, PL/SQL, SQL*Connect, SQL*Forms, SQL*Loader, SQL*Plus, and SQL*Net are trademarks or registered trademarks of Oracle Corporation. All other company or product names mentioned are used for identification purposes only and may be trademarks of their respective owners.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc., in the U.S. and other countries.

Contents

Send Us Your Comments	xiii
Preface.....	xv
1 Introduction	
What is Oracle Portal-to-Go?	1-1
User's View	1-2
How Portal-to-Go Works.....	1-3
Portal-to-Go Content Sources.....	1-4
Portal-to-Go Target Formats.....	1-6
Adding Adapters and Transformers.....	1-6
2 Portal-to-Go Architecture	
Overview	2-1
Portal-to-Go Repository.....	2-2
Configuring Portal-to-Go	2-4
System Properties	2-5
Error Logging	2-8
Development Client Logging.....	2-8
Server Logging.....	2-8
Transaction Logging	2-9
3 Portal-to-Go Tools	
Overview	3-1

Development Tools	3-1
Service Designer.....	3-1
Starting the Service Designer.....	3-2
In-Process Connection	3-3
RMI Connection.....	3-3
Creating Objects.....	3-4
Modifying Objects	3-5
Deleting Objects	3-5
Flagged Objects.....	3-5
Finding Users	3-6
Refreshing the Repository View.....	3-7
Object Identifiers.....	3-7
Web Integration Developer	3-7
XML Editor	3-9
Oracle XML Parser.....	3-9
Portal-to-Go Server Utilities	3-10
LoadXml.....	3-10
Xslt	3-10
CopyObjects.....	3-11
ResetPassword	3-12

4 Walkthroughs

Purpose	4-1
Before You Begin	4-1
Creating a Web Service	4-2
Task 1: Creating the WIDL File in the Web Integration Developer	4-2
Task 2: Publishing the WIDL Interface to the Web Integration Server.....	4-6
Task 3: Starting the Portal-to-Go Service Designer.....	4-7
Task 4: Creating the Master Service	4-7
Task 5: Creating an Alias that Points to the Master Service	4-8
Task 6: Creating a User Group and a User.....	4-9
Task 7: Making the Service Available to a Group.....	4-11
Task 8: Testing the Service	4-12
Accessing the Personalization Portal.....	4-12
Testing the Service on a Phone Simulator.....	4-12

Creating a SQL Service	4-13
Creating Chained Services	4-15
Task 1: Creating the First Service in the Chain	4-15
Generate the WIDL File and Edit the Input Binding	4-16
Edit the Output Binding of the WIDL File	4-16
Task 2: Creating the Second Service in the Chain.....	4-17
Open the Source Page for the Second Service.....	4-17
Generate the WIDL for the Second Service	4-17
Edit the Input and Output Binding of the WIDL File.....	4-18
Edit the URL of the Second Service	4-18
Publish the Service	4-18
Add the Service to Portal to Go.....	4-18
Create the Master Service.....	4-19
Create the Result Transformer	4-19
Create a User and an Alias and Test the Service	4-21

5 Portal-to-Go Services

Overview	5-1
Master Services	5-2
Creating a Master Service.....	5-2
Modifying a Master Service	5-5
General Panel.....	5-5
Init Parameters Panel.....	5-6
Web Integration Init Parameters.....	5-6
SQL Adapter Init Parameters.....	5-7
Stripper Adapter Init Parameters	5-8
URL Adapter Init Parameters.....	5-9
Servlet Adapter Init Parameters.....	5-10
Input Parameters Panel	5-10
Web Integration Input Parameters	5-13
SQL Master Service Input Parameters	5-14
Stripper Adapter Input Parameters.....	5-14
URL Adapter Input Parameters	5-16
Servlet Adapter Input Parameters.....	5-17
Output Parameters Panel	5-18

Result Transformer Panel.....	5-18
Device Transformer Panel.....	5-19
Deleting a Master Service.....	5-19
Creating a Service Using the Servlet Adapter.....	5-19
Service Trees.....	5-21
Creating a Folder.....	5-22
Creating a Service Alias.....	5-24
Creating a Bookmark.....	5-26

6 Managing Users and Groups

User Roles.....	6-1
Portal-to-Go Users.....	6-2
Creating Users.....	6-2
Adding Agent Information to a User.....	6-3
Deleting Users.....	6-6
Modifying Users.....	6-7
User Groups.....	6-7
Creating User Groups.....	6-8
Deleting Groups.....	6-10
Adding Users to a Group.....	6-10
Adding Services to a Group.....	6-10
Removing Users from a Group.....	6-11
Removing Services from a Group.....	6-12
Changing a User's Group.....	6-12
Integrating Users with Existing Provisioning Systems.....	6-12
ProvisioningHook Interface.....	6-13
UserAuthenticationHook Interface.....	6-13

7 Rebranding the Personalization Portal

Overview.....	7-1
Use Sequence.....	7-2
Directory Structure.....	7-6
Login Sequence.....	7-7
Main Page Components.....	7-8
Customization Components.....	7-8

Customizing Services	7-9
Customizing Folders	7-12
Configuring Jobs	7-13
Configuring the User Profile	7-15
Changing Password	7-15
Configuring Agents.....	7-16
Debug Messages	7-17
National Language Support	7-17
Overview.....	7-17
Multibyte Encoding Schemes	7-18
Fixed-width Encoding Schemes	7-18
Variable-width Encoding Schemes.....	7-18
Setting the Multi-Byte Encoding for the Personalization Portal.....	7-18
Localization	7-19
Service Designer	7-19
Personalization Portal.....	7-19

8 Working with Portal-to-Go XML

Why XML?	8-1
Oracle XML Parser	8-2
Portal-to-Go XML Formats	8-3
Adapter Result Format	8-4
Simple Result Format	8-5
Content Model	8-5
Element Reference	8-6
SimpleResult Element.....	8-6
SimpleContainer Element	8-7
SimpleText Element	8-8
SimpleTextItem Element	8-8
SimpleMenu Element.....	8-9
SimpleMenuItem Element	8-9
SimpleForm Element	8-10
SimpleFormItem Element	8-11
SimpleFormSelect Element	8-12
SimpleFormOption Element.....	8-12

SimpleTable Element	8-13
SimpleTableHeader Element	8-13
SimpleTableBody Element	8-14
SimpleRow Element	8-14
SimpleCol Element	8-14
SimpleImage Element	8-15
SimpleBreak Element	8-15
SimplePhone Element	8-15
SimpleEmail Element	8-15
SimpleHref Element	8-16
The Portal-to-Go Repository	8-16
Provisioning DTD	8-17
PanamaObjects Element	8-17
PUSR_LIST Element	8-17
PUSR Element	8-17
Repository XML Reference	8-19
PanamaObjects Element	8-19
PGRP_LIST Element	8-19
PGRP Element	8-20
PUSR_LIST Element	8-20
PUSR Element	8-20
TRAN_LIST Element	8-21
XTRA Element	8-21
EXT_ATTR Element	8-22
JTRA Element	8-22
LDEV_LIST Element	8-22
LDEV Element	8-22
ADAP_LIST Element	8-23
ADAP Element	8-23
PSRV_LIST Element	8-24
FOLD Element	8-24
LINK Element	8-25
MAST Element	8-26
BOMA Element	8-26
AGEN_LIST Element	8-27

AGEN Element	8-27
XML Tools	8-28
XML Editor	8-28
Requirements	8-28
Synopsis	8-28
Using the XML Editor	8-29
LoadXml	8-31
Synopsis	8-31
Options	8-32
Unload Example	8-32
Upload Example	8-33
Upload and Download Utilities	8-33

9 Adapters

What Is a Portal-to-Go Adapter?	9-1
Using Portal-to-Go Adapters	9-2
Implementing Portal-to-Go Adapters	9-2
Content Formats	9-2
Creating Adapters	9-3
Adapter Definition	9-4
Sample Adapter Class	9-4
init()	9-8
invoke()	9-9
getInitDefinition()	9-10
getAdapterDefinition()	9-10
Managing Arguments	9-11
Filtering Adapter Output	9-11
Filtering Adapter Output Sample Code	9-11
Importing an Adapter into the Repository	9-13
Deleting an Adapter	9-14
Modifying Adapters	9-15
Extending the Stripper Adapter	9-15
Naming the Strip Level Class	9-15
Required Methods in the Strip Level Class	9-16
Installing the Strip Level Class	9-16

10 Transformers

What is a Portal-to-Go Transformer?	10-1
Transformers Provided by Portal-to-Go	10-3
Creating Transformers	10-3
Java Transformers	10-4
Sample Java Device Transformer	10-4
SimpleResultToText().....	10-6
format()	10-6
transform().....	10-7
XSLT Stylesheets	10-7
Sample XSLT Device Transformer	10-7
Sample XSLT Result Transformer	10-8
Testing the Transformer	10-9
Managing Transformers with the Service Designer	10-10
Creating a Transformer in the Repository	10-10
Modifying a Transformer	10-11
Removing a Transformer from the Repository	10-11

11 Logical Devices

Overview	11-1
Creating a Logical Device	11-2
Modifying a Logical Device	11-4
Deleting a Logical Device	11-5
Determining the Client Device Type	11-5

12 Integrating Portal-to-Go

Asynchronous Notifications	12-1
Request Queue	12-1
Notification Queue	12-2
Notification Engine	12-2
SMS Listener	12-4
Probe Runtime Monitor and Management	12-5
RMI Server	12-6
FTP Server	12-7

13 Portal-to-Go Extensibility

Overview	13-1
Daemons	13-2
Notification Listeners	13-2
Log Listeners	13-2
Hooks	13-3
Extending the Core Components	13-3

A Simple Result DTD Notes

SimpleResult	A-1
SimpleContainer	A-2
SimpleText	A-3
SimpleTextItem	A-3
SimpleMenu	A-4
SimpleMenuItem	A-4
SimpleForm	A-5
SimpleFormItem	A-6
SimpleFormSelect	A-7
SimpleFormOption	A-8
SimpleTable	A-8
SimpleTableHeader	A-9
SimpleTableBody	A-10
SimpleRow	A-10
SimpleCol	A-11
SimpleImage	A-11
SimpleBreak	A-12
SimplePhone	A-12
SimpleEmail	A-13
SimpleHref	A-13
SimpleHelp	A-14

Glossary

Index

Send Us Your Comments

Portal-to-Go Implementation Guide, Release 1.0.2.2

Part No. A86635-02

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find errors or have suggestions for improvement, please indicate the document name, chapter, section, and page number (if available). You can send comments to:

Oracle Corporation
Oracle Mobile and Embedded Products Documentation
500 Oracle Parkway, Mailstop 4OP6
Redwood Shores, California 94065 USA

If you would like a reply, please provide your:

- Name:
- Title/Company:
- Address:
- Telephone number:
- Email:

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This guide discusses how you can use Portal-to-Go to develop and deliver mobile services. It explains how to manage and use the Portal-to-Go server and development environments. It includes these chapters:

Chapter 1, "Introduction"	Provides an overview of Portal-to-Go.
Chapter 2, "Portal-to-Go Architecture"	Describes the Portal-to-Go architecture and how to configure Portal-to-Go.
Chapter 3, "Portal-to-Go Tools"	Introduces the system development and management tools you can use to implement Portal-to-Go.
Chapter 4, "Walkthroughs"	Provides step-by-step walkthroughs for implementing basic Portal-to-Go services.
Chapter 5, "Portal-to-Go Services"	Describes how to create and manage Portal-to-Go services.
Chapter 6, "Managing Users and Groups"	Describes how to create and manage users and user groups.
Chapter 7, "Rebranding the Personalization Portal"	Describes the Web interface users access to customize their mobile portals.
Chapter 8, "Working with Portal-to-Go XML"	Describes the XML formats Portal-to-Go uses to represent service content and its internal objects.
Chapter 9, "Adapters"	Describes how to create and manage the content acquisition components of Portal-to-Go.

Chapter 10, "Transformers"	Describes the content delivery components of Portal-to-Go.
Chapter 11, "Logical Devices"	Describes how to create and manage the Portal-to-Go objects that represent the target platforms and devices.
Chapter 12, "Integrating Portal-to-Go"	Lists and describes the features that enable you to integrate Portal-to-Go with other systems.
Chapter 13, "Portal-to-Go Extensibility"	Describes how to extend and customize the functional components in the Portal-to-Go architecture.
Appendix A, "Simple Result DTD Notes"	Describes the content of the SimpleResult DTD in a clear format to ensure that the elements, the usage, the attributes, the children, and the DTD declarations are clearly understood.

Introduction

This document introduces Portal-to-Go. Topics include:

- [What is Oracle Portal-to-Go?](#)
- [User's View](#)
- [How Portal-to-Go Works](#)
- [Portal-to-Go Content Sources](#)
- [Portal-to-Go Target Formats](#)
- [Adding Adapters and Transformers](#)

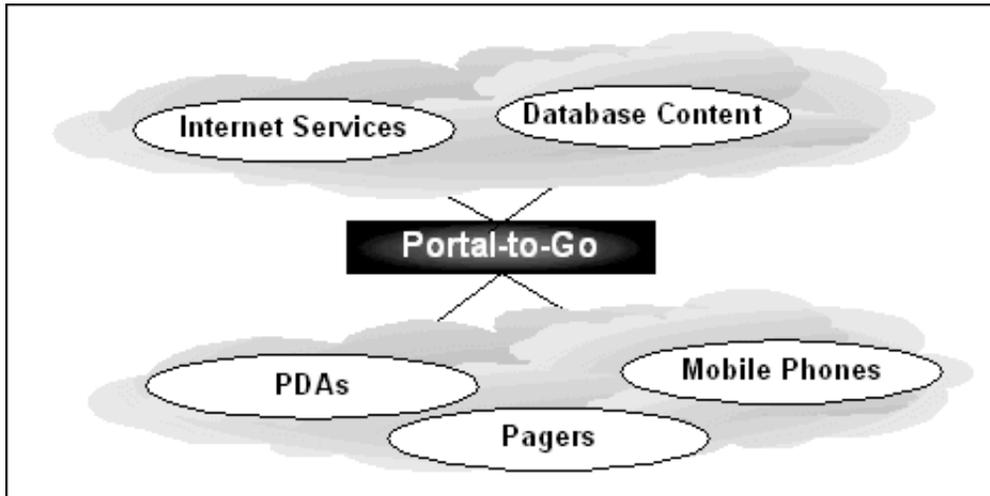
What is Oracle Portal-to-Go?

Oracle Portal-to-Go is a portal service for delivering information and applications to mobile devices. Portal-to-Go makes Web and database applications, such as email, news, and directory services, accessible to mobile device users.

Unlike other solutions, Portal-to-Go does not require you to rewrite content for every target platform. Portal-to-Go uses transformers to dynamically convert content into the format and form factor of the target device. Portal-to-Go provides transformers for a variety of mobile platforms, including smart-phones, Windows CE devices, and Palm Computing Platform devices.

Portal-to-Go works by isolating content acquisition from content delivery. It provides an intermediary format layer, Portal-to-Go XML, between the source

format and the target format. This enables you to easily adapt new content sources for your existing target platforms, or new target platforms for your existing content.



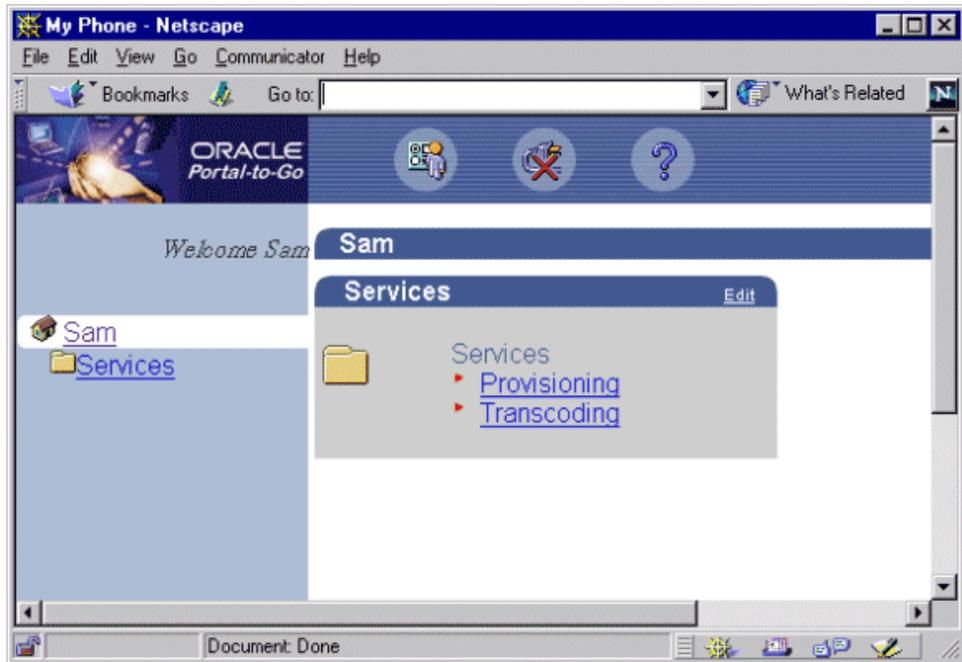
User's View

Portal-to-Go end users access mobile services from the Portal-to-Go device portal. The device portal displays a list of Portal-to-Go-generated services and folders. Users can configure their device portals by selecting the services that they want to access by creating bookmarks and by organizing services in folders.



Users configure their device portals from the Portal-to-Go Personalization Portal. The Personalization Portal is a Web-based interface that users access from their desktop computers.

The Personalization Portal enables users to set frequently entered parameters, such as email addresses, passwords, PINs, and account numbers.



By modifying the elements Portal-to-Go uses to generate the Personalization Portal, you can rebrand the Personalization Portal. Portal-to-Go uses JavaServer Pages, image files, and stylesheets to generate the Personalization Portal.

How Portal-to-Go Works

When a user requests a Portal-to-Go service, Portal-to-Go processes the request as follows:

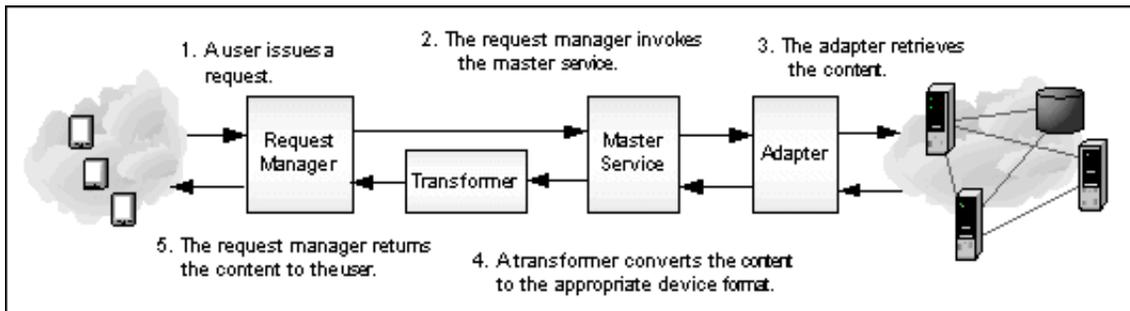
1. The request manager performs user-level preprocessing, including user authentication.

2. If the user is a valid Portal-to-Go user, the request manager creates a request object, which it forwards to the master service associated with the request.
3. The master service invokes a Portal-to-Go adapter to retrieve the requested content. A master service is a Portal-to-Go object that implements a deliverable service. The adapter returns the content in Portal-to-Go XML.

Note: A user request is not the only event that can initiate a Portal-to-Go service; a scheduled job can also initiate a service.

4. A transformer, in the form of an XSLT stylesheet or Java program, converts the content into the format appropriate for the target device.
5. Finally, the request manager returns the information to the user.

The following figure illustrates the Portal-to-Go request and response sequence.



Portal-to-Go Content Sources

Portal-to-Go provides the following pre-built adapters.

Provisioning Adapter

The provisioning adapter enables you to manage Portal-to-Go users from a service interface. It allows you to add, remove, or modify users by invoking a Portal-to-Go service. You must have administrative privileges to use the provisioning adapter.

Additionally, Portal-to-Go provides sample adapters that illustrate how you can build your own adapters for other content sources.

SQL Adapter

The SQL adapter accesses any JDBC-enabled data source. You can use the SQL adapter to create services that query databases, invoke PL/SQL procedures, or call stored procedures in the database.

Servlet Adapter

The Servlet adapter enables you to integrate other applications you might have which are already Java servlets. This provides a convenient way to call them as Portal-to-Go services and make them wirelessly available.

Stripper Adapter

Given a URL, the stripper adapter dynamically retrieves and converts the content of the URL target. Unlike the Web Integration adapter, which uses a predefined mapping of the source content, the stripper adapter dynamically processes the markup tags in the content. Currently, the stripper adapter either removes the original markup tags or leaves them intact. You can extend the stripper adapter, however, so that it processes the tags in another way.

URL Adapter

The URL adapter retrieves the contents of a URL pointing to an XML document that conforms to the SimpleResult DTD. There are no Init Parameters for the URL adapter.

Web Integration Adapter

The Web Integration adapter retrieves and adapts Web content. The Web Integration adapter works with Web Interface Definition Language (WIDL) files to map source content to Portal-to-Go XML. Typically, the source format for the Web Integration adapter is HTML, but you can also use the adapter to retrieve content in other formats, such as XML.

Portal-to-Go provides a visual tool for creating WIDL files, the Web Integration Developer. To create a WIDL file, you identify the elements of a Web page that you want to make accessible to a service. You then associate output and input parameters to the source elements that you want to access in a Portal-to-Go service.

Once you create the WIDL file, you publish it to the Web Integration Server. Publishing a WIDL file to the server makes it accessible to master services in the Portal-to-Go repository. For more information on using WIDLs, see the *Web Integration Developer User's Guide* and *Web Integration Server User's Guide*.

Portal-to-Go Target Formats

Portal-to-Go provides transformers for the following target formats:

WML 1.1

The wireless markup language defined by the WAP Forum. Portal-to-Go provides transformers for a general WML implementation, as well as for device-specific WML implementations.

Tiny HTML

A minimal HTML implementation suitable for handheld devices, such as Palm Computing Platform and Windows CE devices.

VoxML

The Motorola markup language that enables voice interaction with applications.

HDML

The Handheld Devices Markup Language is a simplified version of HTML designed specifically for handheld devices.

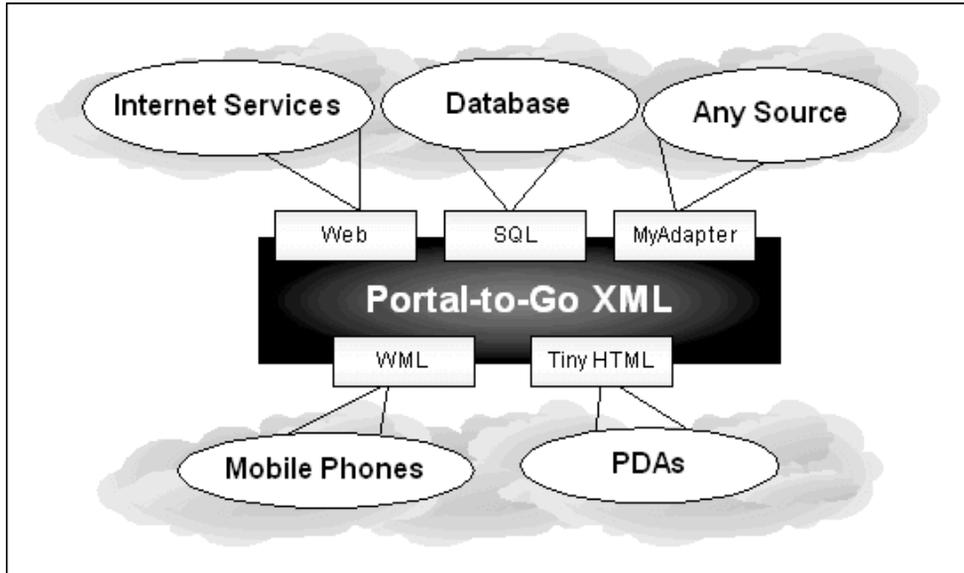
Plain Text

The plain text transformer converts content for Short Message Service-capable devices and email applications.

Adding Adapters and Transformers

You can supplement the adapters and transformers that Portal-to-Go provides with those you create. Portal-to-Go provides tools and interfaces you can use to create your own adapters and transformers. By isolating the content source format from the target format, Portal-to-Go enables you to easily plug in new content sources or target platforms. When you create an adapter for a new content source, or customize a Portal-to-Go adapter, the content is immediately available to all target platforms.

Similarly, when you create a transformer for a new device platform, all your existing content is immediately available to the new target platform.



This provides flexibility for adapting and delivering content, but it also enables you to present information in a device-specific manner. That is, information and applications can be customized for the form factor of each device.

Portal-to-Go Architecture

This document discusses the Portal-to-Go architecture. It introduces the components of Portal-to-Go, and how you can configure them. Topics include:

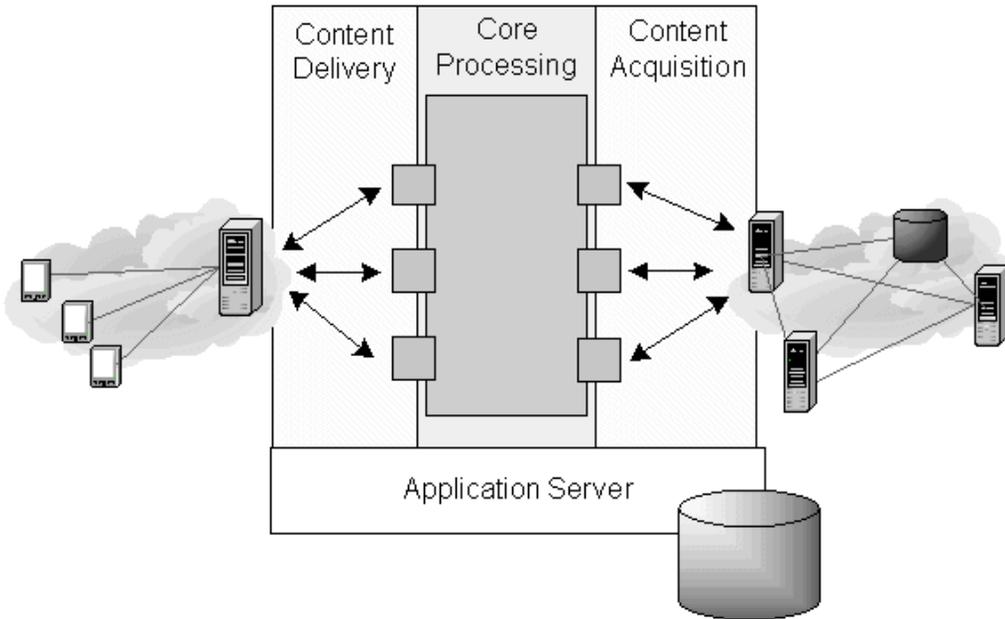
- [Overview](#)
- [Portal-to-Go Repository](#)
- [Configuring Portal-to-Go](#)
- [Error Logging](#)
- [Transaction Logging](#)

Overview

Portal-to-Go consists of server and client components. The client components include tools for developing, managing, and monitoring the Portal-to-Go server. The Portal-to-Go Service Designer, for example, is a visual interface for developing and maintaining Portal-to-Go objects. The Portal-to-Go objects include users, services, adapters, and transformers.

The Portal-to-Go server performs the primary tasks associated with acquiring and delivering mobile services; it receives user requests, and adapts and transforms content.

The Portal-to-Go server has the following architecture for acquiring and delivering content:



Portal-to-Go runs on the Oracle9i Application Server. It uses Oracle8i for its persistent object storage. When you implement Portal-to-Go, you primarily create and manage the repository objects, and establish relationships between them.

Portal-to-Go Repository

The Portal-to-Go repository contains the following types of objects:

Object Type	Description
Adapter	Adapter objects represent the Portal-to-Go interface to content sources. Adapter objects have an attribute called <code>classes</code> . This attribute identifies the archive file that contains the actual Java implementation of the adapter.

Object Type	Description
Master service	The master service object implements the service and invokes a specific adapter. You can associate a custom transformer with a master service. Custom transformers are service- and device-specific. They can be used by only one device and one master service.
Alias	An alias is a pointer to a master service, folder, bookmark, or other alias. You can use aliases to distribute access to services and to override default parameter values defined by the master service. You can make a service accessible to a user by placing an alias to the service in a tree the user owns or can access.
Folder	You can use folders to group services. Folders make services available to end users. Every user has a "home" folder. This contains the services, usually referenced by an alias, that the user can access. A user can also access any service in a folder owned by a group to which the user belongs.
User	Each user object represents a Portal-to-Go end user. The name for the user object must be unique within Portal-to-Go. Users have an external identifier attribute. You can use this ID to associate the Portal-to-Go user with an external provisioning system account. The value of the external ID, for example, may be a telephone number or PIN.
User agent	A user agent identifies a user's device address, such as an email address or telephone number, and device type. This information enables Portal-to-Go to deliver notifications to the user. The user must configure these properties from the Personalization Portal.
User group	User groups help you manage service access for multiple users. You can create user groups based on factors such as subscription level, geographic location, or interests.
Transformer	<p>A transformer converts the content returned by Portal-to-Go adapters. Transformer types include:</p> <ul style="list-style-type: none"> ■ Result transformers, which convert Adapter Result content into SimpleResult content. ■ Device transformers, which convert SimpleResult content into the final target format. <p>A device transformer can be either the default transformer for a logical device, or a custom transformer, which is used to render a specific master service for a specific logical device.</p>
Logical device	A logical device object associates a physical device or an abstract device, such as email, with a transformer.

Object Type	Description
Bookmarks	End users can set bookmarks to external URLs at the Personalization Portal. The bookmark appears as a menu selection in the device portal. Portal-to-Go does not process the content of the URL target. The format of the target content must be supported by the user's device.

Configuring Portal-to-Go

Portal-to-Go supports system-wide extensibility using property files. Each property file stores runtime parameters and processing information for a Portal-to-Go component. Portal-to-Go property files are named with the extension **.properties**.

Property files enable you to plug in new components easily, or extend and configure existing ones. For example, the stripper adapter, which acquires and converts arbitrary Web content, reads configuration settings from the **Strip.properties** file. It uses the settings in the file to locate classes that process the markup tags in content. Each processing class implements a strip level for the adapter. To add a strip level, you create the class that implements the strip level and reference it in **Strip.properties**. This makes the strip level available to the adapter, without requiring you to make changes to the stripper adapter class.

Portal-to-Go includes many property files. The **System.properties** file contains configuration settings for the entire Portal-to-Go system. The following table lists several other property files you can use to configure Portal-to-Go.

Property File	Description
Notification.properties	Contains email and SMS server configuration properties.
Provisioning.properties	Identifies the common root folder for users' service trees.
Rmi.properties	Identifies the server name and listening port of the Portal-to-Go RMI server.
useragent.properties	Links logical devices in the Portal-to-Go repository to the actual user agent parameter received in an HTTP header. Also specifies a default logical device to use if the device type cannot be determined.

These files are located in subdirectories of the following directory:

PortaltoGo_Home/server/classes/oracle/panama/core

System Properties

The Portal-to-Go system configuration file, **System.properties**, contains parameters and component locators for the entire Portal-to-Go system. To start Portal-to-Go, you must have this file in the class package `oracle.panama.core.admin`. This package must be in the environment `CLASSPATH`.

The **System.properties** file contains the following parameters:

Key	Type	Customizable?	Description
version	String	N	The Portal-to-Go version.
locator.boot.check	Boolean	N	If set to true, Portal-to-Go performs a bootstrap check procedure during system boot. The default is true.
db.connect.string	String	Y	A valid connect string, including user name and password. The precise format depends on the driver you use.
db.driver	String	Y	The JDBC driver Portal-to-Go uses to access the repository. Possible values are THIN, OCI8, INTERNAL, and CUSTOM. If set to CUSTOM, you must also set the <code>db.driver.class</code> parameter.
db.session.min	Integer	Y	The minimum or optimal number of open database sessions. This parameter is used to tune the session pool. It can increase when necessary, but always attempts to return to the specified number of concurrent open database sessions.
db.schema	String	Y	The repository schema to use if not using the default.
db.driver.class	Class name	Y	The class that implements the JDBC driver if using a custom driver. The default is the Oracle JDBC driver.

Key	Type	Customizable?	Description
name.separator	Character	N	The separator used in the service path. This cannot be modified after installation.
locator.objectcache.class	Class Name	N	The object cache implementation class.
objectcache.check.timetolive	Seconds	Y	The time-to-live, in seconds, of a persistent object. After this time Portal-to-Go reconstructs the object.
objectcache.check.interval	Seconds	Y	The time required for the cache monitor to check the cache. If set to -1, Portal-to-Go does not invoke the cache monitor and the cache is not cleared.
session.expiration.time	Seconds	Y	The time-to-live attribute of a session.
session.expiration.checkinterval	Seconds	Y	The time required for the session monitor to check an open session.
locator.persistent.class	Class name	N	The locator implementation for persistent objects.
locator.logger.class	Class name	N	The logger implementation.
locale.language	String	Y	The language used. This is represented by the ISO code for the representation of names of languages. The default is English.
locale.country	String	Y	The location. This uses the ISO 3166 country code. The default is United States.
log.level	String list	Y	The log level. This can contain any of the following: error, warning, notify, or transaction.
log.directory	Path	Y	The log file directory.
log.file.name	File name	Y	The log file name pattern.
log.file.maxsize	Integer	Y	The maximum number of log records in the same file.

Key	Type	Customizable?	Description
log.tx.directory	Path	Y	The transaction log file directory.
log.tx.file.name	File name	Y	The transaction log file name pattern.
log.tx.file.maxsize	Integer	Y	The maximum number of transaction entries in the same file.
log.tx.file.pattern	Formatted string	Y	The pattern used to write transaction log records.
log.console	Boolean	Y	Determines whether true log output is written to <code>stderr</code> .
algorithm.password	Java algorithm name	Y	The hash algorithm for passwords.
locator.external.hook.class	Class name	Y	The default implementation of <code>UserAuthenticationHook</code> .
locator.useragent.class	Class name	Y	The default implementation of the device recognition class.
locator.transform.class	Class name	Y	The default transformer implementation class.
locator.request.daemon.class	Class name list	Y	The system daemons to start.
locator.request.queue.class	Class name	N	The asynchronous request queue implementation.
locator.notification.queue.class	Class name	N	The asynchronous notification queue implementation.
locator.notification.dispatcher.class	Class name	Y	The default notification engine implementation.
locator.postprocessor.class	Class name	N	The transformer post-processor implementation.
locator.provisioning.class	Class name	Y	The default implementation of the provisioning hook.
system.java.protocol.handler.pkgs	Class name	Y	The Java class package that provides SSL (Secure Sockets Layer) support.

Key	Type	Customizable?	Description
order.services	String	Y	The sorting order for Portal-to-Go services, folders, and bookmarks on the output devices.

Error Logging

Portal-to-Go generates logging information for both the server and development client components.

Development Client Logging

By default, the Service Designer writes error information to the log directory of your Service Designer home directory. You can modify this setting, and the log naming pattern, in the **preferences.xml** file, which is located in the ServiceDesigner/resources directory.

Server Logging

Portal-to-Go writes server error information to the files and directory specified in the **System.properties** file. You can open the log files directly, or view them from the Portal-to-Go Runtime Information and Management interface (the Probe interface) at port 8090. Click the "Files" link.

Portal-to-Go specifies these types of errors:

Error	Value
ERR_SERVICE_UNAVAILABLE	PTG-001
ERR_SERVICE_ERROR	PTG-002
ERR_SERVICE_NOT_FOUND	PTG-003
ERR_CONFIGURATION_ERROR	PTG-004
ERR_INTERNAL_ERROR	PTG-005
ERR_DATABASE_ERROR	PTG-006

In addition to error messages, Portal-to-Go provides extensive runtime exception logging. When fatal exceptions occur, Portal-to-Go logs the exceptions and stack

traces in the global log file. If you need to contact Oracle Support Services, you should have the log information available.

Transaction Logging

The transaction log is a log file on the Portal-to-Go server that provides information regarding user access. To enable transaction logging, you must include transaction logging as a log level in the **System.properties** file. For example:

```
log.level=Warning, Error, Notify, Transaction
```

You can also use the **System.properties** file to set the directory to which Portal-to-Go writes transaction logs, and to specify a file name pattern for the log files. You can also configure the transaction record pattern.

If you enable transaction logging, when a user accesses a service that has a cost value set, the transaction log generates data. You can set a cost for a master service using the Portal-to-Go Service Designer. You can access the transaction log data programmatically, to generate billing information or to integrate Portal-to-Go transaction information with an external billing management system.

The following sample shows the log results of four stock quote queries by a single user. The sample uses the default logging pattern, which you can modify in the **System.properties** file. It lists the time of access, the user name (Sample), the service, and the price per access (25):

```
[11/14/99 4:03:26 PM] Sample /master/OnlineQuoteYahoo 25  
[11/14/99 4:03:29 PM] Sample /master/OnlineQuoteYahoo 25  
[11/14/99 4:03:35 PM] Sample /master/OnlineQuoteYahoo 25  
[11/14/99 4:04:43 PM] Sample /master/OnlineQuoteYahoo 25
```

Portal-to-Go Tools

This document describes the tools you can use to develop and maintain Portal-to-Go. Topics include:

- [Overview](#)
- [Development Tools](#)
- [Portal-to-Go Server Utilities](#)

Overview

Portal-to-Go provides tools to help you create, manage, and deliver mobile services. These tools include visual interfaces for developing and managing repository objects, and utilities for managing the server and deploying Portal-to-Go.

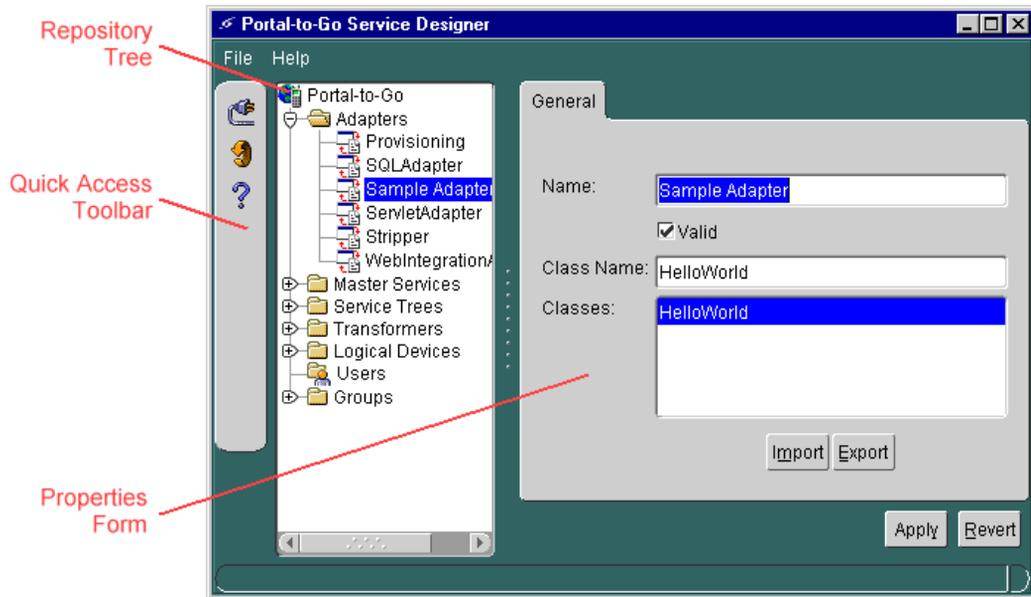
Development Tools

Portal-to-Go provides the following visual development tools:

- [Service Designer](#)
- [Web Integration Developer](#)
- [XML Editor](#)

Service Designer

The Service Designer is a visual interface for implementing and managing Portal-to-Go. You use the Service Designer to create and modify Portal-to-Go objects, including users, adapters, transformers, and services.



The Service Designer provides a tree view of the Portal-to-Go repository. The tree shows Portal-to-Go object classes, such as adapters and transformers, as folders, or branch nodes. It shows instances of those classes, such as the WML 1.1 transformer, as objects, or leaf nodes. For performance reasons, the default tree view in the Service Designer does not display more than 100 objects of any type. This, however, can be configured in the **ptgsd.properties** file.

The following sections describe basic tasks you can perform in the Portal-to-Go Service Designer.

Starting the Service Designer

When you start the Service Designer, the Log In dialog appears. The Log In dialog includes fields for the location, the Portal-to-Go user name, and password. In the location field, you specify the URI (Universal Resource Identifier) of the repository that you want to develop. You can connect to the repository either directly (called in-process communication) or through the Portal-to-Go server (using RMI). The value you enter in the Location field determines the connection mode.

Note: The in-process connection is deprecated. You should use RMI.

In-Process Connection To connect directly to the repository, you must include the JDBC connect string of the database repository in the **System.properties** file on the development machine. Specify the connect string as the value of the `db.connect.string` property. Then, when logging into the repository from the Service Designer, you enter "ptgdb://" in the Location field. The Service Designer ignores any text that follows this string.

Important: The Service Designer has its own **System.properties** file. When the Service Designer connects to the server that Portal-to-Go is installed on, it looks for the **System.properties** file that is in the directory structure of Portal-to-Go. Therefore, you should set the path so that the Service Designer goes to that file and not to its own **System.properties** file. This requires that you have the Portal-to-Go server classes in the JVM used by the Service Designer. In this case, the Service Designer still connects to the repository directly using JDBC protocol. The JDBC connect string is stored in the **System.properties** file and cannot be changed from the Service Designer user interface.

The other option is to delete the Service Designer's own **System.properties** files (on the server and the client) and replace each with a copy of the Portal-to-Go **System.properties** file that is in the directory structure of Portal-to-Go on the server.

RMI Connection To connect to the repository through the Portal-to-Go server, the RMI listener must be running on the Portal-to-Go server. By default, the RMI listener is started when you start the Portal-to-Go server.

In the Location field of the Log In dialog, enter the URI of the Portal-to-Go server in any of the following formats:

- `rmi://hostname:port/servername`
- `rmi://hostname:port`

- *rmi://hostname*
- *rmi://:port/servername*
- *rmi://:port*
- *//hostname:port/servername*
- *//hostname:port*
- *//hostname*
- *//:port/servername*
- *//:port*
- *hostname:port/servername*
- *hostname:port*
- *hostname*
- *:port/servername*
- *:port*

The default port number and server name are:

- Port: 2008
- Server name: PanamaServer

You can modify these defaults in the **oracle.panama.core.admin.Rmi.properties** file. The **System.properties** file identifies the default RMI listener used as follows:

```
locator.request.daemon.classes=oracle.panama.core.rmi.server.ServerImpl
```

Creating Objects

You can create an object in the Service Designer as follows:

1. In the repository tree, highlight the class node of the type of object that you want to create, or the folder in which you want to create the object.
2. Right-click the class or folder. A pop-up menu appears with menu items for creating objects.
3. Click the appropriate menu item for creating the new object.

The Service Designer then presents a form, or a sequence of forms, that lets you configure the object. You can navigate through the forms by clicking the Next or

Previous button. When you finish configuring the new object, click the Finish button. This creates the object in the repository.

Modifying Objects

When you highlight an object in the repository tree, the right panel displays the object's properties. If an object has more than one property panel, you can navigate between panels by clicking the tabs at the top of the panel.

You can modify an object by making changes directly to the properties in the panel. To save your changes, click the Apply button. This saves the changes you have made to any of the object's panels, not just the current panel.

To cancel the changes you have made to an object since last saving, click the Revert button. This restores the object to its prior state. If you make changes to an object, but click on another object in the repository before clicking Apply, the changes to the object do not take effect.

Deleting Objects

To remove an object from the repository:

1. Highlight the object in the tree view and right-click.
2. In the pop-up menu, select the menu item for deleting the object.
3. Confirm the action.

You cannot delete an adapter used by a master service. If you delete a master service that is referenced by aliases, Service Designer deletes the aliases as well.

Flagged Objects

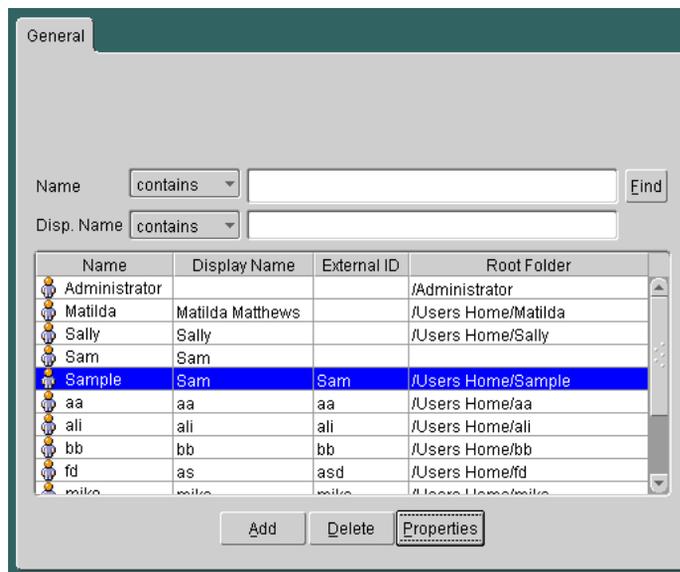
Flagged objects appear in red type in the repository tree. Portal-to-Go flags objects that are specified as not valid or not visible. Specifying an object as not valid or not

visible prevents its use. You may choose to disable an object, for example, while it is under development or testing.



Finding Users

To modify a user in the Portal-to-Go repository, you must first return the user as a result of a search query. The user query form, which appears when you select Users in the repository tree, lets you query users by name or external ID. The external ID is a user's unique identifier, such as an account or telephone number, in an external provisioning system. Clicking the Find button with the search field empty returns all users.



You can create, delete, or modify users by highlighting the user and clicking the Add, Delete, or Properties button.

Refreshing the Repository View

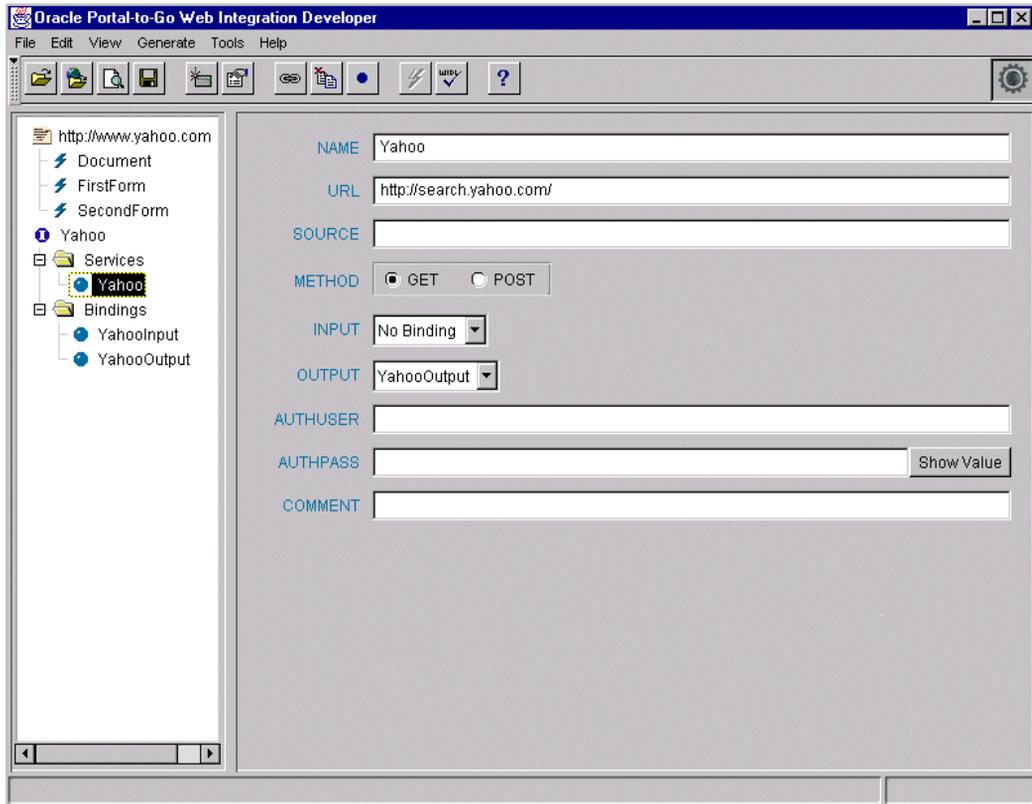
The Service Designer caches object information. If multiple Service Designer users work in the same repository concurrently, the object view for each user may not be up-to-date with the state of the repository. To retrieve the latest state of the repository, click the refresh icon on the toolbar.

Object Identifiers

Each object in the Portal-to-Go repository is identified by a unique object ID. You can see the object ID for an object by moving the mouse over the object. The object ID helps you to identify the object when you access the repository with another tool, such as the Portal-to-Go XML Editor.

Web Integration Developer

The Web Integration Developer is a visual interface for creating and testing WIDL services. The Web Integration adapter uses WIDL services to map Web content. You also use the Web Integration Developer to publish the WIDL interfaces to the Web Integration Server.

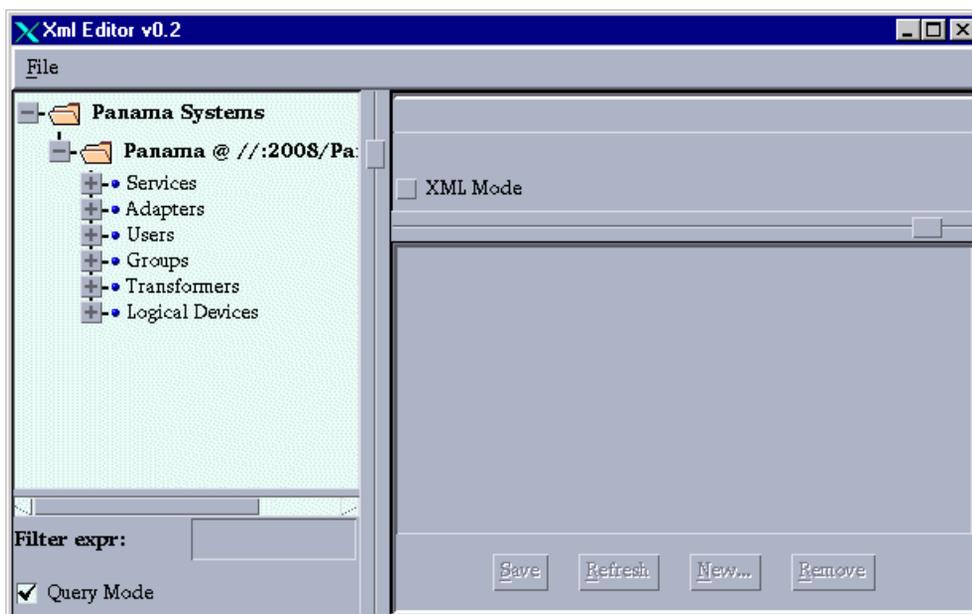


When you create a WIDL service, you parameterize a source page. You create input parameters for input elements, such as text fields and selection lists, and output parameters for returned content. The Web Integration adapter renders the Web content in one of the Portal-to-Go result formats. It converts the content according to the rules defined in the adapter. For example, record variables, which you can create in the Web Integration Developer, are data structures that assemble related variables in a group. The Web Integration adapter converts record variables into menu items, and assumes that the first element in a record is a linked name and the second is the target URL.

For more information on creating WIDL services and interfaces, see the *Portal-to-Go Web Integration Developer User's Guide*.

XML Editor

The Portal-to-Go XML Editor is a text editor that allows you to create, modify, and delete repository objects as XML elements. You should not attempt to use the XML Editor unless you understand the XML structure of the Portal-to-Go repository. Portal-to-Go does not validate the changes you make with the XML Editor. You should always back up your repository before modifying it with the XML Editor.



For more information on using the XML Editor, see ["XML Editor"](#) in [Chapter 8](#), ["Working with Portal-to-Go XML"](#).

Oracle XML Parser

Portal-to-Go processes XML documents using the Oracle XML Version 2 parser. The parser supports the Document Object Model (DOM) and the Simple API for XML (SAX) interfaces. It supplements the DOM interface with extensions, which are detailed in the *Portal-to-Go API Specification*.

Portal-to-Go Server Utilities

Portal-to-Go provides these utilities for managing and deploying Portal-to-Go:

- [LoadXml](#)
- [Xslt](#)
- [CopyObjects](#)
- [ResetPassword](#)

LoadXml

The `LoadXML` utility allows you to import and export the Portal-to-Go repository as an XML file. The `LoadXML` utility makes it easier for you to deploy developed repositories. For more information on using the `LoadXML` utility, see "[LoadXml](#)" in [Chapter 8, "Working with Portal-to-Go XML"](#).

Xslt

The `Xslt` utility is a tool you can use to test stylesheets. `Xslt` uses the XML DOM parser and the XSL processor included with the Oracle XML processor. You can use `Xslt` to apply the stylesheets you create to any XML document.

`Xslt`, which is a command-line utility, reads from standard input and writes its results to standard output.

Synopsis

```
oracle.panama.util.Xslt [stylesheet]
```

Options

The `Xslt` utility takes the following option:

Option	Description
<code>stylesheet</code>	The XSL stylesheet that the utility applies to the XML document.

Xslt Example

```
java oracle.panama.util.Xslt mystylesheet.xml < myxml.xml
```

CopyObjects

The `CopyObjects` utility allows you to copy services from one Portal-to-Go site to one or more other sites. You can use this utility, for example, to deploy services from a testing and development environment to production servers.

`CopyObjects` sends services to target servers as XML elements. It takes a folder object as a command-line argument. To copy a service, therefore, you must first place the service in a folder. When invoked, `CopyObjects` copies the folder and all its contents to the target Portal-to-Go sites you specify at the command-line.

Requirements

All source and target Portal-to-Go servers must have the Portal-to-Go RMI server running. For more information, see ["RMI Server"](#) in [Chapter 12, "Integrating Portal-to-Go"](#).

The basic configuration of adapters and transformers must be the same on the source and target servers. `CopyObjects` only sends services, links, and folders. It does not send configuration data.

Synopsis

```
oracle.panama.core.util.CopyObjects [-f folder] [-s source ][targets...]
```

Options

The `CopyObjects` utility takes the following options:

Option	Description
<code>-f <i>folder</i></code>	The name of an existing folder in the source system.
<code>-s <i>source</i></code>	The source Portal-to-Go server. Specify the source in the format: <code>username/password@//hostname:port/servername</code>
<code><i>targets</i></code>	The target Portal-to-Go servers. Specify targets in the format: <code>username/password@//hostname:port/servername</code>

CopyObjects Example

This example copies a folder and its content from a local server to production sites at `m1` and `m2`.

```
java oracle.panama.core.util.CopyObjects \
    -f /master/finance \
    -s user/pw@//:2008/PanamaServer \
    user/pw@/m1:2008/PanamaServer \
```

```
user/pw@/m2:2008/PanamaServer
```

ResetPassword

The `ResetPassword` utility allows you to change a user's password. You can use this utility to recreate a user's password if it is forgotten. `ResetPassword` does not perform an authentication check.

Synopsis

```
oracle.panama.core.util.ResetPassword [username ][new_password ]
```

Options

The `ResetPassword` utility takes the following options:

Option	Description
<i>username</i>	The name of an existing user.
<i>new_password</i>	The new password for the user.

ResetPassword Example

This example resets the password of the user Scott to tiger.

```
java oracle.panama.core.util.ResetPassword scott tiger
```

Walkthroughs

This document provides walkthroughs for implementing Portal-to-Go services. Topics include:

- [Purpose](#)
- [Before You Begin](#)
- [Creating a Web Service](#)
- [Creating a SQL Service](#)
- [Creating Chained Services](#)

Purpose

The following walkthroughs provide step-by-step instructions for creating and distributing mobile services with Portal-to-Go. In the walkthroughs, you create services based on the Web Integration adapter and the SQL adapter. At the end of each walkthrough, you test your development using a device emulator.

These walkthroughs familiarize you with the Portal-to-Go tools, including the Web Integration Developer and the Service Designer, and how they enable you to create mobile services.

Before You Begin

Before you begin the walkthroughs, make sure that you correctly install and configure the Portal-to-Go server. Also, you must complete the client installation on your development environment.

Creating a Web Service

In this walkthrough, you create a Portal-to-Go Web service. First, you use the Web Integration Developer to map the elements of a sample Web page, `quote.yahoo.com`, to input and output parameters. You then use the Service Designer to create a Portal-to-Go service that retrieves stock quotes based on the mapping.

For more information on using the Web Integration Developer, see the *Portal-to-Go Web Integration Developer User's Guide*.

This walkthrough guides you through the following tasks:

- [Task 1: Creating the WIDL File in the Web Integration Developer](#)
- [Task 2: Publishing the WIDL Interface to the Web Integration Server](#)
- [Task 3: Starting the Portal-to-Go Service Designer](#)
- [Task 4: Creating the Master Service](#)
- [Task 5: Creating an Alias that Points to the Master Service](#)
- [Task 6: Creating a User Group and a User](#)
- [Task 7: Making the Service Available to a Group](#)
- [Task 8: Testing the Service](#)

Task 1: Creating the WIDL File in the Web Integration Developer

Follow these steps to create a Web page mapping, or Web Integration Definition Language (WIDL) file, using the Web Integration Developer.

Open the Source Page

Open the source page in the Web Integration Developer as follows:

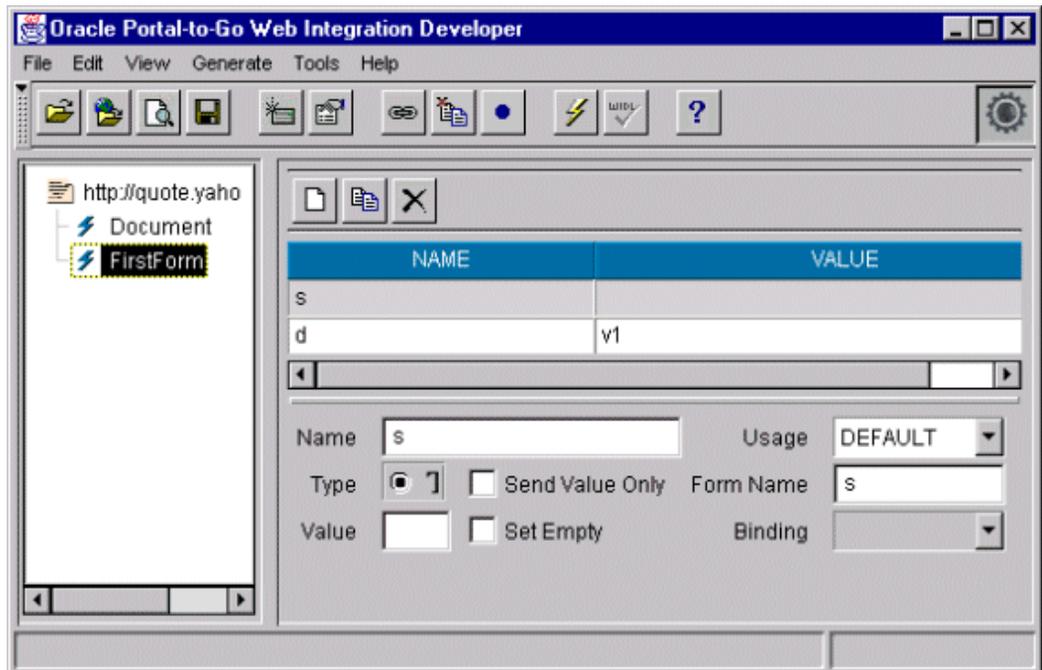
1. Start the Oracle Web Integration Developer.
2. Select Open URL from the File menu or toolbar.
3. In the Open URL dialog box, type the following URL:

`http://quote.yahoo.com`

4. Click OK.

The Web Integration Developer retrieves the page, parses it, and adds it to the Document Browser (in the left frame). For this example, the Document Browser shows the following items:

- The first item (<http://quote.yahoo.com>) identifies the open document.
- The second item (the Document node) represents the contents of the HTML document: paragraphs, images, links, lists, and tables.
- The third item (the FirstForm node) represents the form in this document. When you open a page that contains forms, the Web Integration Developer creates a form node for each form in the document.



Generate a WIDL File

Next, generate a WIDL file from the source page as follows:

1. Select FirstForm in the left frame.
2. Select WIDL from the Generate menu.

3. Complete the New Service dialog as follows:

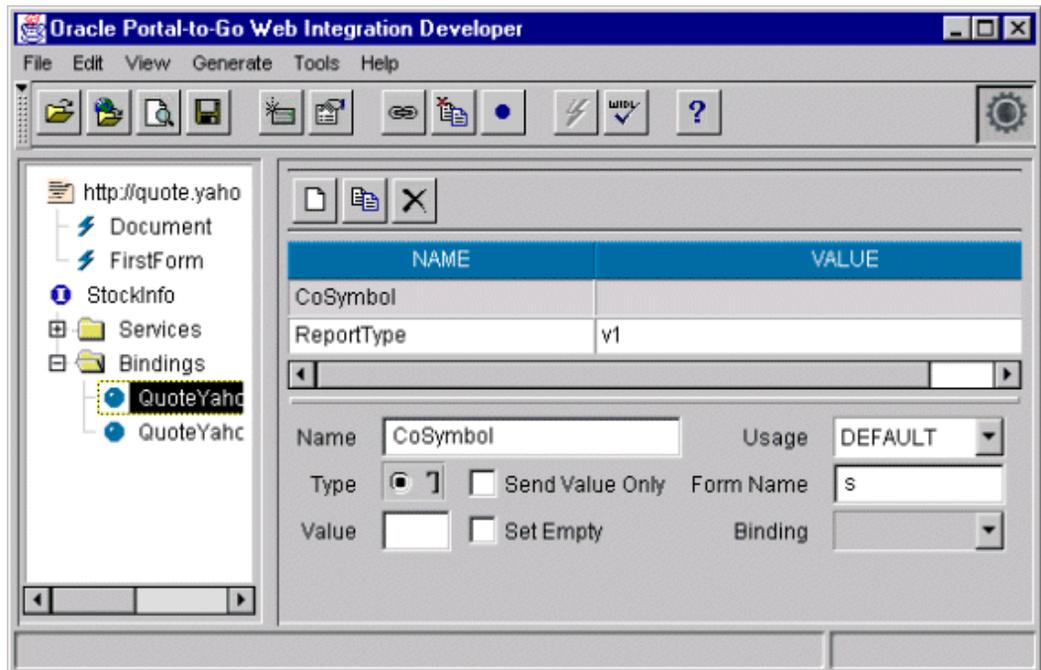
In this Field...	Type...
Interface	StockInfo
Service	QuoteYahoo_GetQuote

4. Click OK.
5. In the Generate New WIDL for Service dialog, type `ORCL`.
6. Click the Submit button.

Edit the Input Binding of the WIDL File

The input variables defined in this WIDL service have the same names as those specified in the HTML form. You can edit the input binding to make these names more meaningful as follows:

1. Expand the Bindings folder in the left frame.
2. Click the `QuoteYahoo_GetQuoteInput` binding.
3. Click the variable “d” in the variable list in the right frame. In the Name field, type `ReportType` and press Enter.
4. Click the variable “s” in the variable list. In the Name field, type `CoSymbol` and press Enter.



Edit the Output Binding of the WIDL File

The output defined by this service extracts all elements from the document returned by the Web page. To extract just the stock quotes to pass back to the client application, you can edit the output binding:

1. Select the QuoteYahoo_GetQuoteOutput binding in the left frame.
2. In the right frame, scroll through the variable list until you reach the table variables.
3. Click the variable table4. Make sure the content of this variable is the stock price of ORCL. If not, click the variable whose content is ORCL's stock price.
4. On the Sample tab, select the cell that contains the current stock price.
5. Click the right mouse button and select Create New Variable From Selection from the pop-up menu.
6. Type `CurrentPrice` in the New Variable dialog and click OK. The `CurrentPrice` variable now appears in the variable list.

7. Delete all other variables from the variable list.
8. Save the WIDL file.

Test the WIDL File

To test the WIDL file in the Web Integration Developer:

1. Click QuoteYahoo_GetQuote in the Services folder in the left frame.
2. Select Test Service from the Tools menu.
3. Type any valid stock symbol in the CoSymbol field.

Task 2: Publishing the WIDL Interface to the Web Integration Server

Publishing a WIDL interface makes it accessible to Web Integration services that you create in the Service Designer.

You must have administrator authority on the Web Integration Server to perform this procedure. When you publish an interface, the services in that interface are added to those already on the Web Integration Server. If you create a service with the same name as an existing service, the existing service is overwritten.

Follow these steps to publish your WIDL file (StockInfo) to the Web Integration Server.

1. Select StockInfo in the left frame.
2. From the File menu, select Publishing, then Publish Interface.
3. In the Specify Server field in the Publish Interface dialog, type the name of the Web Integration Server to which you want to publish this interface. Specify the server name using the format:
host_name:port
4. The Web Integration Server uses packages to organize services. You can click Update Packages to view a list of packages on the specified server, then add the service to a specific package. In this case, however, you can add the sample service to the Default package. Click OK.
5. If the User Name and Password dialog appears, enter a user name and password for the selected server. This user must have administrative privileges. Click OK.

The Web Integration Developer copies the interface to the selected package on the Web Integration Server and notifies you that the interface is successfully published.

Assuming the package is enabled, the services in this interface are immediately available for execution.

Task 3: Starting the Portal-to-Go Service Designer

After creating the WIDL service, you create a Portal-to-Go master service based on the WIDL service. You can create Portal-to-Go services using the Service Designer. The Service Designer is a visual tool for creating and managing Portal-to-Go objects.

Start the Service Designer

To start the Portal-to-Go Service Designer, from the Start menu, point to the Oracle for Windows NT menu item. Click Portal-to-Go, then Service Designer.

Connect to the Portal-to-Go Repository

When the Service Designer starts, the Log In dialog appears. Use the Log In dialog to connect the Service Designer to the Portal-to-Go repository that you want to develop. The Log In dialog enables you to select a server name that you have already used, or to enter a new server name.

To connect to the Portal-to-Go repository from the Log In dialog:

1. In the location field, enter the name of the Portal-to-Go server. Then enter your user name and password in the appropriate fields.
2. Click OK. The Portal-to-Go server objects appear in the tree view on the left.

Note: The Service Designer displays an Error dialog for invalid entries. By clicking the Show Details button, you can view the Java runtime exception details.

Task 4: Creating the Master Service

Using the Service Designer, create the master service for the StockInfo WIDL file by following these steps:

1. Right-click the Master Services folder in the Portal-to-Go repository tree view.
2. Click Create New Master Service to display the Create New Master Service form sequence. Complete the form as follows:
 - a. In the Name field, type `Stock Quotes`. You can name the new master service that you are creating anything you want. In this example, it is named Stock Quotes.

- b. Click the Browse button next to the Adapter field. Select `WebIntegrationAdapter` and click OK.
 - c. Select the Valid check box.
 - d. Select the Visible check box.
 3. Click Finish. The new master service Stock Quotes now appears under the Master Services folder in the Portal-to-Go repository tree view.
 4. Click the Init Parameters tab in the right frame. Complete this form as follows:
 - a. For the Web Integration Server, type the machine name of the Web Integration Server in the format: *host_name.domain:port*
 - b. For the Interface field, type `StockInfo`. This is the name of the WIDL interface you created in ["Task 1: Creating the WIDL File in the Web Integration Developer."](#)
 - c. Do not supply a value for the `WIDL_FILE` parameter.
 5. Click Apply. Then click Yes in the dialog asking you if you want to apply the changes. The Service Designer populates the Input Parameters tab with the input parameters for the adapter.
 6. Click the Input Parameter tab and select the User Customizable check box for the `CoSymbol` parameter.
 7. Click Apply. Then click Yes in the dialog asking you if you want to apply the changes.

This creates the master service Stock Quotes in the Portal-to-Go repository. You can now create aliases to make the master service available to users.

Task 5: Creating an Alias that Points to the Master Service

To add a master service alias in the service tree, you first create a new folder in the service tree and then add an alias to the master service in this folder. Follow these steps to create an alias for the master service Stock Quote in the service tree:

1. Right-click Service Trees in the Portal-to-Go repository tree view.
2. Click Create New Folder to display the Create New Folder form.
3. Complete this form as follows:
 - a. In the Name field, type `My Folder`.
 - b. Select the Valid check box.

- c. Select the Visible check box.
4. Click Finish. My Folder now appears under Service Trees in the Portal-to-Go repository tree.
5. Right-click My Folder.
6. Click Create New Alias to display the Create New Alias form.
7. Complete this form as follows:
 - a. In the Name field, enter `My Service`.
 - b. Click the Browse button next to the Service field. A Browse Services dialog appears. Expand the Master Services folder within the Browse Services dialog and select Stock Quotes. Click OK.
 - c. Select the Valid check box.
 - d. Select the Visible check box.
8. Click Finish.
9. Click the Input Parameters tab in the right frame and select the User Customizable check box for `CoSymbol`.

This creates an alias object in the Portal-to-Go repository. Aliases, like master services, have input parameters. You can use the input parameters of an alias to override any input values specified by the master service. For this walkthrough, you use the default settings provided by the master service.

Task 6: Creating a User Group and a User

Now you create a user group and then a user.

First, you create an overall folder which will later contain a unique "root folder" for each user that you create. Each "root folder" will have the same name that you used to name the user. You can call the overall folder anything you like that is meaningful to you, keeping in mind that this folder will contain the root folders for all the users. In our example, the overall folder is called "Users Home." Create this folder in the Service Trees node of the repository tree in the left frame.

The root folder of a particular user contains the services (including aliases, master services, and folders) which the specified user can access.

One benefit is that if you have many users, you can create an overall folder for each grouping of users. For example, using the label "Users Home" as an illustration, you can create "Users Home A" for all users whose name starts with A, "Users Home B"

for users whose name starts with B, and so on. In this fashion, grouping the users' root folders within higher-level folders minimizes the number of objects that are displayed in the Service Designer's tree view. (For performance reasons, the default view in the Service Designer's tree view does not display more than 100 objects of any type.)

Second, you create a user group.

Finally, you create the user and add the user to a group.

Create the Overall Folder for the Users' Root Folders

To create the overall folder for the users' root folders (using "Users Home" as an example):

1. Right-click Service Trees in the repository tree view.
2. Click Create New Folder to display the Create New Folder form.
3. Complete this form as follows:
 - a. In the Name field, enter `Users Home`.
 - b. Select the Valid check box.
 - c. Select the Visible check box.
4. Click Finish.

The folder named "Users Home" now appears under Service Trees in the repository tree.

Create the User Group

To create a user group:

1. Select the Groups folder in the Portal-to-Go repository tree.
2. Right-click and select Create New Group.
3. In the Name field of the Create New Group dialog, type `MySampleGroup`.
4. Click Finish. Because there are no changes to make at this point, do not click Apply when the panel for the General tab appears.

This creates a user group called `MySampleGroup` in the repository.

Create the User and the User's Root Folder

To create the user:

1. Select the Users folder in the Portal-to-Go repository tree.
2. Right-click and select Create New User.
3. Configure the user as follows:
 - a. In the Name field, type `Sam`.
 - b. For the Display Name, type `Sam`.
 - c. For the Password, type `welcome`.
 - d. For the Password Hint, type `welcome`.
 - e. Click the Browse button next to the User Root Parent field and select the folder "Users Home," then click OK.

This action creates a **user folder** named "Sam" under the overall folder named "User's Home." You will see the user folder for Sam when you expand the User's Home node in the tree view.
 - f. Click the down arrow on the list box for the list of groups and select `MySampleGroup`, which you had created earlier.
 - g. Select the Enabled check box.
4. Click Finish to create the user in the Portal-to-Go repository.

Task 7: Making the Service Available to a Group

To make the service you created available to a group of users:

1. Expand the Groups folder in the Portal-to-Go repository tree view.
2. Click `MySampleGroup`. The parameters for the group appear in the right frame.
3. Click the Service Folders tab in the right frame to display the Service Folders panel. The Service Folders panel includes two fields, Service Folders and Selected Service Folders.
4. Under Service Folders, expand the Services folder.
5. Click My Folder.
6. Click the right arrow (>) button to move My Folder to the Selected Folders field.
7. Click Apply.
8. Click Yes in the Confirm Apply Changes dialog.

This makes the service available to all users in the group MySampleGroup. In ["Task 8: Testing the Service"](#), you test this service as the user Sam.

Task 8: Testing the Service

In this walkthrough, you test your Portal-to-Go service using the Personalization Portal and a phone simulator. For more information on the Personalization Portal, see [Chapter 7, "Rebranding the Personalization Portal"](#).

Accessing the Personalization Portal

Log on to the Personalization Portal to view the service you created. To access the Personalization Portal:

1. From a Web browser, enter the following URL:

```
http://PTGServer.domain/panama/server/papz/login.jsp
```

2. Enter the following for the user name and password:

In this Field...	Type...
Username	Sam (Note: Sam is case sensitive)
Password	welcome

The services available to the user Sam appear in the left frame.

3. Click My Folder, then My Service.
4. Click on QuoteYahoo GetQuote.
5. Enter a valid stock symbol and click Submit.

Portal-to-Go retrieves and displays the current price of the stock.

Testing the Service on a Phone Simulator

You can use any WML- or HDML-based phone simulator to test the service. If you do not already have one, download and install a simulator. If you are working

behind a firewall, you must configure the proxy server settings in the simulator before using it to access external sites.

Note: Phone.com provides a WML-enabled phone simulator in the UP.SDK, a software development kit for creating WML and HDML services. The development kit is freely available from the Phone.com Web site.

Follow these steps to test the service from a phone simulator.

1. Start the simulator.
2. Enter the following URL in the Go window:

```
http://PTGServer.domain/panama/server/papz/alias for the  
oracle.panama.ParmImpl servlet
```

This is the URL of the device portal for your Portal-to-Go installation.

3. Log in using Sam and welcome as the user name and password. The preconfigured Portal-to-Go services for the user Sam appear.
4. Select My Folder, then My Service.
5. Enter a valid stock symbol and click OK.

Note: If the phone simulator returns an HTTP error, you should refresh the cookie cache and source cache. Go to the phone simulator install directory and refresh (or delete) the files **CookieCache** and **SourceCache**.

Portal-to-Go retrieves and displays the current price of the stock.

Creating a SQL Service

In this walkthrough, you create a Portal-to-Go service based on the SQL adapter. You create a service that checks the current time from the system table DUAL and displays it to the user.

Before you begin the walkthrough, make sure you can access a Portal-to-Go server from your development environment. Start the Service Designer, as described in Step 1 of the Web Integration walkthrough, and follow these steps:

1. Right-click the Master Services folder in the Portal-to-Go repository tree view.
2. Click Create New Master Service to display the Create New Master Service form sequence. Complete this form as follows:
 - a. For the Name field, enter `SQL Time`.
 - b. For the adapter, click Browse. Select `SQLAdapter` and click OK.
 - c. Select the Valid check box.
 - d. Select the Visible check box.
3. Click Finish. The new master service `SQL Time` appears under the Master Services folder in the Portal-to-Go repository tree view.
4. Click the Init Properties tab in the right frame. Complete this form as follows:
 - a. For the SQLTYPE, type `Query`.
 - b. For the STATEMENT, type:

```
select sysdate from dual
```
 - c. For the JDBC_CONNECT_STRING, enter the JDBC connect string for the database on which to query, as follows:

```
jdbc:oracle:thin:@host_name:port:SID
```
 - d. For the PASSWORD and USERNAME, type the password and user name of any database user. You can use the sample user `Scott` with the password `tiger`.
5. Click Apply.

Note: For more information on creating master services, see [Chapter 5, "Portal-to-Go Services"](#).

You can now test the service from a simulator, as described in steps 5 through 8 of the Web Integration service walkthrough.

Creating Chained Services

Chained services link services to each other in a sequential way so that each chained service is dependent for input on the service that immediately precedes it. Chained services are needed for Web sites which require the user to interact with a Web page to access the rest of the content on the Web site. For example, the first Web page prompts the user for a username and password before the user can access the form on the second page, which then produces the information on the third page. Chaining one service (Service A) to another service (Service B) works in such a way that when the Output Binding of Service A completes successfully, it chains to the Input Binding of Service B.

The chained service you build in this walkthrough illustrates a typical use of chained services. Based on the Yahoo finance site, the chained service you build in this section allows the user to enter a stock ticker symbol, receive the current stock quote, select details, and receive detailed information about the stock, such as a chart. The first service requests that the user enter a stock ticker symbol. It returns the current quote for that stock. When the user clicks on Details and selects Charts, the second service returns the chart for that stock.

You should become familiar with a site you want to base a service on by first opening it in a browser. Opening the site and viewing its HTML source code lets you become familiar with the structure of a site. In a browser, open the Yahoo site for getting stock quotes:

```
http://quote.yahoo.com
```

The query form on this page takes a stock's ticker symbol. It returns that stock's current quote.

This walkthrough assumes that you are familiar with the basic steps required to create a Web service in the Oracle Web Integration Developer. If you are not, complete the Web Integration walkthrough in "[Creating a Web Service](#)" before attempting to follow this walkthrough.

This walkthrough does not include error checking. Usually, you would add error catching logic to the service.

Task 1: Creating the First Service in the Chain

The overall interface name for the chained services in this walkthrough is YahooFinance. The first service in the chain sequence is called **Quote**. This service prompts the user for a stock ticker symbol (such as, ORCL) and returns the stock

quote. The following steps provide a basic method for creating the first service in the chain.

Generate the WIDL File and Edit the Input Binding

1. In the Oracle Web Integration Developer, open the following URL:

`http://quote.yahoo.com`

Two nodes appear in the document browser, Document and FirstForm.

2. Highlight the FirstForm node and click Generate, then WIDL. Complete the New Service dialog as follows and click OK:

In this Field...	Type...
Interface	YahooFinance
Service	Quote

3. In the Generate a New WIDL for Service "Quote" dialog box, enter the stock symbol ORCL. Click Submit.
4. Expand the bindings node and highlight the QuoteInput binding by selecting it. The variables in the Quote service's input binding appear in the edit pane of the Variables tab.
5. Rename the variable s to s1, and d to d1.

Edit the Output Binding of the WIDL File

1. Select the QuoteOutput binding. In the variables list, highlight the row for table4 by clicking on it. The Sample tab shows the values in table4. You can create new variables based on these values.
2. In the Sample tab, highlight the ORCL cell. Right click. Select **Create new variable from Selection**. Name the new variable **symbol**.
3. Similarly, highlight the cell that shows the ORCL stock price (in the table4 sample result). Right click. Select **Create new variable from Selection**. Name the new variable **price**.
4. Follow the same procedure as in the above two steps to create the new variable **change** from the cell that shows the value for change.

5. Delete all output variables from QuoteOutput, except for the three new output variables (symbol, price, change) that you just created in steps 7, 8, and 9 above.
6. Save the WIDL file.
7. Test the first service, Quote, with other stock symbols.

Task 2: Creating the Second Service in the Chain

This walkthrough takes you through the steps to build the second service, named **Detail**, in the chained service YahooFinance. While you can create chained services using separate interfaces, in this walkthrough you add the second service to the YahooFinance interface.

The Detail service takes the stock quote as input and returns stock charts.

Create the second service as follows:

Open the Source Page for the Second Service

1. Open the following page in a browser:
`http://finance.yahoo.com`
2. Enter ORCL as the quote symbol and select Chart as the type of search (instead of Basic).
3. Click Search. As you see, a chart search returns more information than a basic search. The second service in the chain returns this detailed information.
4. Select the URL in your browser's Location field,
`http://finance.yahoo.com/q?s=orcl&d=b`, and open this URL in the Web Integration Developer.

Generate the WIDL for the Second Service

1. Select FirstForm in the left frame.
2. Select WIDL from the Generate menu.
3. Complete the New Service dialog as follows:

In this Field...	Type...
Interface	YahooFinance
Service	Detail

4. Click OK.
5. In the Generate a New Widl for Service "Detail" dialog box, enter the stock symbol ORCL. Click the Submit button.

Edit the Input and Output Binding of the WIDL File

1. Expand the Bindings folder in the left frame.
2. Select DetailInput. Rename the variable `s` to `s2`, and `d` to `d2`.
3. Select DetailOutput. In the variables list, highlight `table4`.
4. Find the cell with the Open price. Highlight the cell, right click and create a new variable from selection. Name the new variable **opened**.
5. From the day's range cell, create a new variable named **range**.
6. Similarly, from the cell showing volume, create a new variable named **volume**.
7. From the cell showing the 52-week range, create a new variable named **annualrange**.
8. Create a new variable named **pe** from the cell that shows the P/E value.
9. Create a new variable named **mktpcap** from the cell that says Mkt cap.
10. Delete all variables other than the ones that you just created in steps 9 to 14 above.

Edit the URL of the Second Service

1. Highlight **Detail** under the Services node.
2. Remove `q?s=orcl&d=b` from the Source URL.
3. Save the WIDL file.

Publish the Service

Publish the new service to the WIDL server, as described in "[Task 2: Publishing the WIDL Interface to the Web Integration Server](#)."

Add the Service to Portal to Go

The new Master Service is `YahooFinance`. It points to the interface `YahooFinance`; it includes the two chained services **Quote** and **Detail**. To add this service to the Portal-to-Go repository, start the Service Designer and follow these steps:

Create the Master Service

1. Right click the Master Services node in the repository tree and choose Create new Master Service.
 - a. Type YahooFinance as the service name.
 - b. Browse for Adapter, and select WebIntegrationAdapter
 - c. Select Valid and Visible. Click Finish.
2. In the new service, click the Init Parameters tab.
 - a. For the WebIntegrationServer, specify the URL of the Web Integration Server to which you published the Web Integration interface, in the form: *machine.domain:port*
 - b. For the interface, type YahooFinance and click Apply.
3. Click the Input Parameters tab.
 - a. Make s1 User Customizable.
 - b. Click the Value field of the PAsession variable. A drop-down arrow should appear. Click the arrow and choose Quote.
 - c. For the variable OutputType, select a value of RawResult.
4. Click the Add button. Click in the name field of the new input variable and type PAdebug. Set the value to 1.
5. Deselect the User Customizable check box for PAsession, if set.
6. Click Apply.

Create the Result Transformer

1. Click the Result Transformer tab.
2. Click the Add button. In the name field, delete the default name, and enter Quote. Click the Edit button, and paste the following style sheet in the XSL Editor window and click OK. Be sure to replace the existing template text in the window.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:apply-templates
select="ServiceRequest/Result/AdapterResult"></xsl:apply-templates>
  </xsl:template>
  <xsl:template match="AdapterResult">
```

```

<SimpleResult>
  <SimpleMenu>
    <SimpleMenuItem>
      <xsl:attribute
name="target">__REQUEST_NAME__?PAservicepath=__SERVICE_URL_ENC__
&#38;PAsection=Detail&#38;s2=<xsl:value-of
select="symbol"></xsl:value-of></xsl:attribute>
      <xsl:value-of select="symbol"></xsl:value-of>
      <!-- this xsl:text is just for beautifying the output -->
      <xsl:text> - </xsl:text>
      <xsl:value-of select="price"></xsl:value-of>
      <!-- this xsl:text is just for beautifying the output -->
      <xsl:text> - </xsl:text>
      <xsl:value-of select="change"></xsl:value-of>
    </SimpleMenuItem>
  </SimpleMenu>
</SimpleResult>
</xsl:template>
</xsl:stylesheet>

```

3. Click the Add button. A name field for a new stylesheet appears with a default name.
4. Replace the default name with Detail. Click the Edit button.
5. Replace the existing template with the following and click OK:

```

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:apply-templates
select="ServiceRequest/Result/AdapterResult"></xsl:apply-templates>
  </xsl:template>
  <xsl:template match="AdapterResult">
    <SimpleResult>
      <SimpleContainer>
        <SimpleText>
          <SimpleTextItem>
            <xsl:value-of select="opened"></xsl:value-of>
          </SimpleTextItem>
          <SimpleTextItem>
            <xsl:value-of select="range"></xsl:value-of>
          </SimpleTextItem>
          <SimpleTextItem>
            <xsl:value-of select="volume"></xsl:value-of>
          </SimpleTextItem>
          <SimpleTextItem>

```

```
        <xsl:value-of select="annualrange"></xsl:value-of>
    </SimpleTextItem>
    <SimpleTextItem>
        <xsl:value-of select="pe"></xsl:value-of>
    </SimpleTextItem>
    <SimpleTextItem>
        <xsl:value-of select="mktcap"></xsl:value-of>
    </SimpleTextItem>
</SimpleText>
</SimpleContainer>
</SimpleResult>
</xsl:template>
</xsl:stylesheet>
```

Create a User and an Alias and Test the Service

1. Create a new user and a new alias as described in the first walkthrough.
2. Test the service from the Personalization portal.

Portal-to-Go Services

This document describes how to create and manage Portal-to-Go services. Topics include:

- [Overview](#)
- [Master Services](#)
- [Creating a Service Using the Servlet Adapter](#)
- [Service Trees](#)

Overview

Services enable end users to access the functionality of Portal-to-Go adapters. Services represent the link between the content source and the delivery target. They tie a specific adapter to the logical devices in the Portal-to-Go repository.

Services also define how users access the adapter. They can restrict or grant user access to the adapter's parameters. Services can set default values for a parameter, they can enable users to set a default, or they can hide the parameter from users altogether.

The following repository objects encapsulate and manage Portal-to-Go services:

- Master services
- Service aliases
- Folders (or service trees)
- Bookmarks

Master services provide the actual implementation of the service. They specify the adapter used for the service and any service-specific parameters. An alias is a

pointer to the service. Aliases are useful for distributing access to a master service among many users and groups. End users can access a master service only if the master service, or an alias to the master service, appears in a service tree they own or that a group they are a member of owns. Aliases can set parameter values that override values set at the master service.

You can manage service objects (master services, aliases, folders, and bookmarks) using the Service Designer. The Service Designer provides a tree view of the Portal-to-Go repository. The repository tree includes a Master Services folder and a Service Trees folder for managing services. You can create service objects in either folder. In most cases, however, you should place folders and master services in the Master Services root folder, and user- and group-owned folders and aliases in the Service Trees root folder.

Master Services

By mapping an adapter to device transformers, master services link Portal-to-Go content sources to the delivery platforms. Each master service is based on one adapter. A master service creates its own instance of the adapter it uses. Therefore, several services can use the same type of adapter, and each can pass its own service-specific argument values.

Portal-to-Go end users typically invoke a master service by clicking on a menu item from their device. The information returned by the master service can be text, such as a movie review, or an application, such as an airline booking system. While a user request usually invokes a master service, a scheduled job can also invoke a master service.

Creating a Master Service

You use the Create New Master Service form to create a master service in the Portal-to-Go repository. This form enables you to create a minimally configured master service object in the repository. After you complete the form, you must finish configuring the master service using its property panels.

This section describes the Create New Master Service form. "[Modifying a Master Service](#)" describes the additional parameters you must configure to create a master service.

To display the Create New Master Service form:

1. Right-click Master Services or Service Trees in the Portal-to-Go repository tree view.

2. Click Create New Master Service.

The Create New Master Service form appears.

This form allows you to configure a new master service's general parameters. It takes the following parameters:

Parameter	Value
Name	The name of the master service.
Adapter	The name of the adapter that the master service uses. To select an adapter, click the Browse button and select from the available adapters.
Valid	Select the Valid check box to enable the master service.
Visible	Select the Visible check box to make the master service accessible to users.
Owner	The user or group that owns the master service. Only an owner can modify the master service. Typically, this is a Portal-to-Go administrator or designer.

Parameter	Value
Sequence	<p>This integer value lets you alter the order in which services and folders appear on output devices. By default, these appear in order by sequence number, then name. You can enter values in the sequence fields to rearrange the order in which the services and folders appear.</p> <p>By default, Portal-to-Go sorts services and folders in ascending order by sequence number, then by name. You can change this behavior by altering the <code>order.services</code> property in the System.properties file. You can specify sorting by sequence number, name, or date last written, in either ascending or descending order.</p>
Area ID	<p>An integer value that allows you to specify whether a service is visible based on a user's physical location. This enables you to implement location-specific services. This is an optional value.</p> <p>To use this parameter, you implement the <code>requestBegin()</code> method in the <code>ParmHook</code> (request manager) interface. <code>requestBegin()</code> is called every time Portal-to-Go receives a user request. You can place logic in this method that invokes an external positioning system. <code>requestBegin()</code> then passes the result in the request object. When Portal-to-Go invokes the service, it calls <code>isRuntimeVisible()</code>, passing it the user's position. You can test the position against the Area ID configured for the master service in <code>isRuntimeVisible()</code> to determine whether to make the service available.</p>
Cost	<p>The cost per service access. If you set a value for this parameter, and configure the System.properties file to enable transaction logging, Portal-to-Go writes billing information to a transaction log. This is an optional value.</p>
Description	<p>An optional description of the master service.</p>

Click the Finish button to create the master service in the repository. Complete the configuration as follows:

1. Highlight the new master service object in the repository tree view. Click the Init Parameters tab.
2. Enter the initialization parameters for the master service. The parameters vary depending on which type of adapter you are using. If creating a service based on the Web Integration adapter, you specify the host name and port number of the Web Integration Server to which you have published your WIDL interface, and the name of the WIDL interface. Click Apply.

3. The Service Designer populates the Input Parameters panel with the parameters defined generally for the adapter and specifically for the interface. In the Input Parameters panel, configure the parameters for the master service, specifying caption, user access, format, and values. Remove the Mandatory check box selection if the parameter can be empty.
4. The Output Parameters panel lets you specify a caption for an output parameter. By default, Portal-to-Go uses the parameter name for the output caption. This panel lets you override the default.
5. If creating chained services, create a result transformer for each service in the interface.
6. Click Apply to save your changes.

The following section provides more information on the master service forms.

Modifying a Master Service

To modify a master service using the Service Designer, click the master service that you want to change. The object's property tabs appear in the right-hand panel of the Service Designer. You can enter new values directly in the parameter fields, then click Apply to save your changes.

Master services have the following property panels:

- General
- Init Parameters
- Input Parameters
- Output Parameters
- Result Transformer
- Device Transformer

General Panel

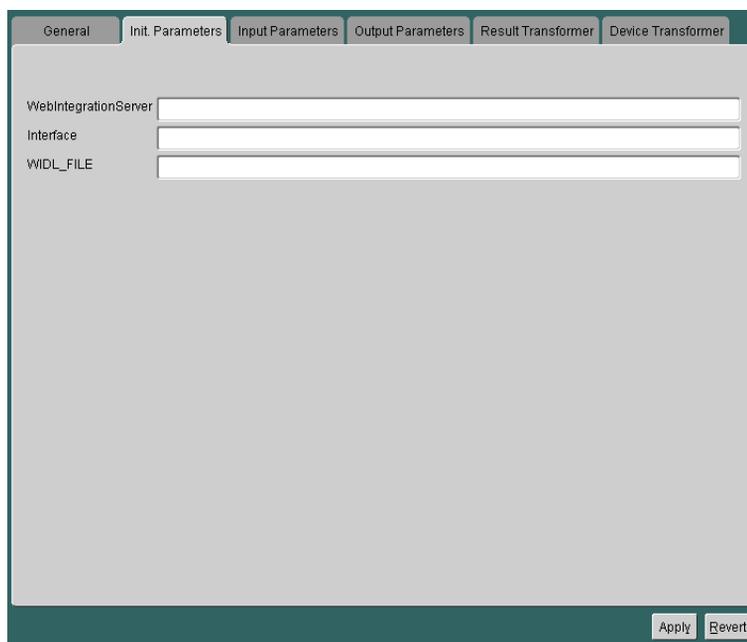
The General panel contains parameters that identify and describe the master service. It includes fields for the service name, the adapter on which the service is based, and the cost of the service. Typically, you set the properties in this panel using the Create New Master Service form when first creating the master service. For more information on the General properties, see "[Creating a Master Service](#)".

Init Parameters Panel

The Init Parameters panel shows the initialization parameters for the adapter. These parameters vary depending on the adapter implementation. The Service Designer generates the fields in this panel from the adapter definition for the master service. When Portal-to-Go first invokes the adapter, it passes the values you set in this panel to the adapter.

The following sections describe the initialization panels for master services based on the adapters provided by Portal-to-Go.

Web Integration Init Parameters For a master service based on the Web Integration adapter, the Init Parameters panel appears as follows:



The screenshot shows a software interface with a dark green header bar containing six tabs: "General", "Init. Parameters", "Input Parameters", "Output Parameters", "Result Transformer", and "Device Transformer". The "Init. Parameters" tab is selected and highlighted. Below the tabs is a light gray area with three text input fields. The first field is labeled "WebIntegrationServer", the second "Interface", and the third "WIDL_FILE". At the bottom right of the panel, there are two buttons: "Apply" and "Revert".

The Init Parameters panel contains the following parameters:

Parameter	Value
WebIntegrationServer	The machine name and listening port of the Web Integration Server. If the Web Integration Server and the Portal-to-Go server reside on the same machine, use <code>localhost:port</code> . This field is required. The server you specify in this field must be running for the Service Designer to return the adapter parameters.
Interface	The WIDL interface name. This interface must be published to the Web Integration Server. You can publish the interface using the Web Integration Developer. You cannot currently use the WIDL_FILE parameter to identify a WIDL service.
WIDL_FILE	Do not enter a value for this parameter.

SQL Adapter Init Parameters For a master service based on the SQL adapter, the Init Parameters panel appears as follows:

The screenshot shows a software interface with a tabbed menu at the top. The tabs are: General, Init. Parameters, Input Parameters, Output Parameters, Result Transformer, and Device Transformer. The 'Init. Parameters' tab is selected. Below the tabs is a large text area labeled 'The Statement'. Below that is a dropdown menu labeled 'Type of Statement'. Below the dropdown are four text input fields labeled 'Password', 'Username', 'JDBC Driver', and 'JDBC Connect String'. At the bottom right of the panel are two buttons: 'Apply' and 'Revert'.

The panel includes the following parameters:

Parameter	Value
The Statement	<p>The actual SQL statement that invokes the query, PL/SQL procedure, or stored procedure.</p> <p>Note: The SQL statement should be entered without a semicolon.</p> <p>You can use input variables in the SQL statement. You must indicate a variable in the statement by prefixing the variable with a colon. For example, you can specify an input variable in a PL/SQL statement as follows:</p> <pre>begin mypackage.foo(:expr); end;</pre> <p>Where <code>:expr</code> is the name of the variable. You must define the parameter manually in the input panel.</p>
Type of Statement	<p>The type of SQL statement used by the master service. Allowable values:</p> <p>QUERY: for a select statement. This type of statement returns a Simple Result document. You can use output filtering with QUERY statements. For information on filtering output, see "Filtering Adapter Output" in Chapter 9, "Adapters".</p> <p>PLSQL: to use a PL/SQL procedure. This type of statement returns results to a database buffer.</p> <p>CALL: to run a stored procedure (SQL92 syntax only). This returns either a Simple Result or an Adapter Result element.</p>
Password	The password of the database user.
Username	The name of the database user.
JDBC Driver	The type of JDBC driver used to access the SQL data source.
JDBC Connect String	The database connect string for the database to access. For example, to access an Oracle database using the thin driver, use the connect string: <code>jdbc:oracle:thin:@domain:port:SID</code>

Stripper Adapter Init Parameters For a master service based on the stripper adapter, the Init Parameters panel appears as follows:

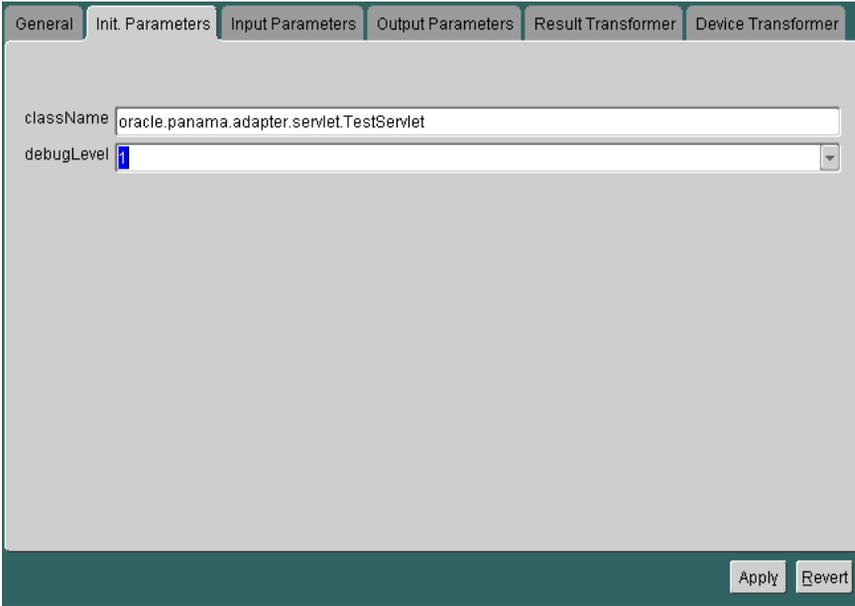
The panel contains the following parameters:

Parameter	Value
config	The resource file that contains the stripper adapter properties. The properties file contains proxy settings, the strip classes available to the adapter, and a translation table for special characters. To use the strip adapter, you must set the proxy in the configuration file. The default is oracle.panama.adapter.stripper.Strip.
debugLevel	The debug level. Possible values are: 0, for none; 1, for notifications; and 2 for trace.
maxResultLength	The maximum number of characters allowed in the result. You use this value to limit the size of the returned page. The default is 15360. The adapter truncates the result if it is longer than the maximum length.

For more information on using the Stripper adapter, see ["Extending the Stripper Adapter"](#) in [Chapter 9, "Adapters"](#).

URL Adapter Init Parameters There are no Init parameters for this adapter.

Servlet Adapter Init Parameters For a master service based on the servlet adapter, the Init Parameters panel appears as follows:



The panel contains the following parameters:

Parameter	Value
className	The complete class name of the Java servlet file. For example: oracle.panama.adapter.servlet.TestServlet.
debugLevel	The debug level. Possible values are: 0, for none; 1, for notifications; and 2 for trace.

For more information on using the Servlet adapter, see ["Creating a Service Using the Servlet Adapter"](#) in this chapter.

Input Parameters Panel

The Input Parameters panel displays the input parameters for the adapter. The Service Designer queries the adapter definition to determine the parameters that appear in this panel. The master service passes the input parameter values to the adapter's invoke method every time the adapter executes.

Some parameters rely on user input for values. The values for other parameters, such as name of the WIDL service in the WIDL interface (`PAsection`), are set by the master service or master service alias. `PAsection` is an internal parameter, not exposed to the end user. In addition to `PAsection`, Portal-to-Go provides these input parameters:

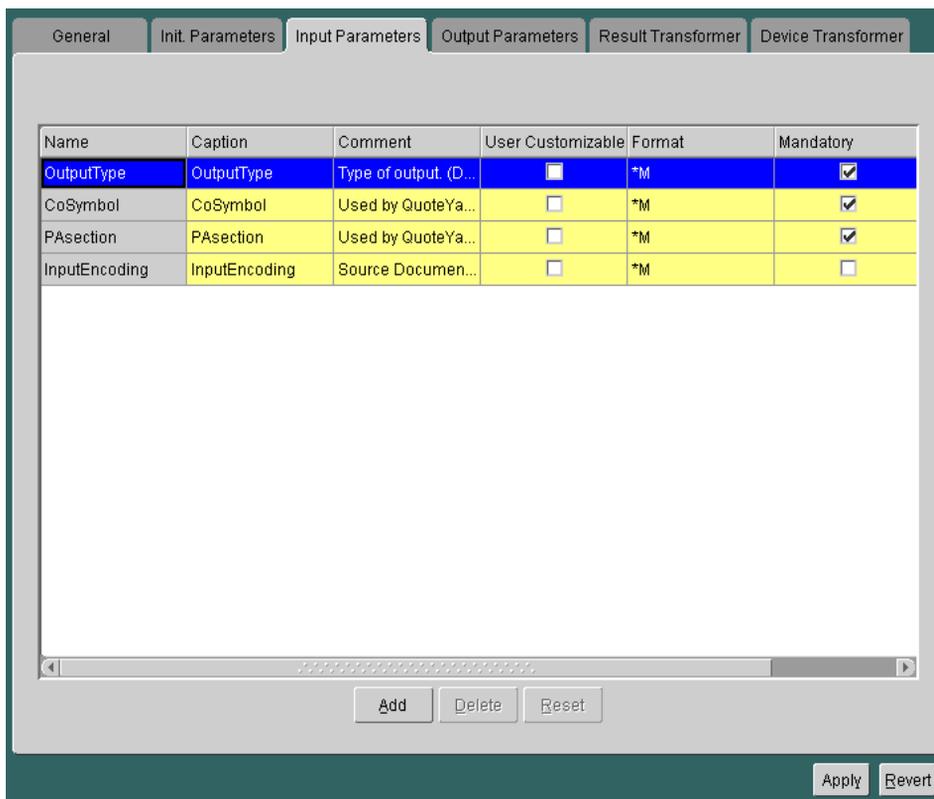
Variable	Value
<code>PAservicepath</code>	The relative path to a Portal-to-Go service, <code>/UsersFolders/joe/myChain</code> , for example.
<code>PAdebug</code>	The debugging option. If true (set to 1), Portal-to-Go produces verbose output to the log files. In this case, in addition to notifications and warnings, Portal-to-Go writes the results of adapter invocations to the log file. This enables you to examine service content in its internal, XML format, which can help you to create result transformers and solve service and transformer problems.
<code>PAsection</code>	The WIDL adapter uses this value to identify the service that serves as the entry point in the chained service sequence.
<code>PAuserid</code>	The user name.
<code>PApassword</code>	The user password.
<code>PAid</code>	The Portal-to-Go session identifier.

You can configure your parameters in the Service Designer. Every Portal-to-Go parameter has the following attributes:

Parameter	Value
Name	The name of the input parameter. The Service Designer sets the name of the input parameter by querying the adapter definition.
Caption	The caption is the label that Portal-to-Go uses for the parameter when prompting for user input.

Parameter	Value
Comment	<p>In the case of master services based on the Web Integration adapter, Portal-to-Go automatically populates this cell with the name of the WIDL service that uses the parameter.</p> <p>For services based on other adapters, you can use this column to document the parameter. The comment is only used internally.</p>
User Customizable	<p>Specifies whether the end user can set a value for this parameter at the Personalization Portal. You can make most input parameters customizable by the user. In particular, you should set this option for parameters that may be difficult for a user to enter from a mobile device. This includes email addresses and personal identification numbers.</p>
Format	<p>This mask sets the expected data entry mode for the user device. For example, if you expect the user to enter numbers for the parameter, you use the format code N. This works only with WML 1.1-compliant devices.</p> <p>The default format is *M. Other formats include:</p> <ul style="list-style-type: none">■ A, for entry of uppercase letters or punctuation■ a, for entry of lowercase letters or punctuation■ N, for entry of numbers.■ X, for entry of uppercase letters.■ x, for entry of lowercase letters. <p>For a complete list of formats, see the <i>Wireless Application Protocol Wireless Markup Language Specification, Version 1.1</i>.</p>
Mandatory	<p>Select this check box if this parameter must have a value. Remove the selection for optional parameters.</p>
Value	<p>For most parameters, this value represents the default value for the parameter. If you specify a default value, Portal-to-Go does not prompt the user for a value. Default values can be overridden by a value specified by a service alias or, if the parameter is visible to the user, by the user at the Personalization Portal.</p> <p>The <code>PAsection</code> parameter is used by the Web Integration adapter. For <code>PAsection</code>, this value is the name of the WIDL service that the Web service should use. You can select the names from a drop-down selection list. If you do not specify a value for <code>PAsection</code>, the Portal-to-Go service includes all WIDL services in the WIDL interface.</p>

Web Integration Input Parameters The following figure shows a sample Inputs panel for a Web service.



The master service determines the parameters to display in the panel by querying the adapter. Every input parameter defined in the WIDL interface appears in the Inputs panel, including parameters for other WIDL services within the WIDL interface.

In addition to the custom input parameters that you create, Web Integration services provide these parameters:

- OutputType
- PAsection
- InputEncoding

The `OutputType` specifies the type of XML output that the adapter should return. You can specify `RawResult`, to return content in Adapter Result format, or `SimpleResult`, to return content in Simple Result format. If returning raw result format, you must create a result transformer that converts the result into Simple Result for the device transformer. The result transformer should have the same name as the value you use for the `PASection` parameter; that is, it should have the same name as the WIDL service. You use `RawResult` for chained services.

`PASection` is the name of the WIDL service that you want the master service to invoke. A WIDL interface can include more than one WIDL service. Portal-to-Go lists the WIDL service names in a selection list in the value field.

`InputEncoding` specifies the encoding used to encode the source document. The source document is the URL that was used to create the WIDL file for this service. See [Chapter 4, "Walkthroughs"](#), for information on creating a Web service. The default value of this parameter is UTF-8. If the language of the source document is an Asian language, you can change the default encoding to the appropriate multi-byte encoding according to the IANA standards for the particular Asian language that is used in the source document. The `InputEncoding` parameter enables you to specify or change the encoding. It is part of the multi-byte character support. ["National Language Support"](#), in [Chapter 7, "Rebranding the Personalization Portal"](#), provides more information about multi-byte character support.

SQL Master Service Input Parameters You can configure SQL input parameters just as you can Web service parameters. You specify input parameters in the SQL statement you use to implement the service. For information on creating parameterized SQL services, see ["SQL Adapter Init Parameters"](#).

Stripper Adapter Input Parameters For a master service based on the stripper adapter, the Inputs panel appears as follows:

Name	Caption	Comment	User Customizable	Format	Mandatory	Value
url	url	Target URL	<input type="checkbox"/>	*M	<input checked="" type="checkbox"/>	
beginTag	beginTag	Match begin string	<input type="checkbox"/>	*M	<input checked="" type="checkbox"/>	
endTag	endTag	Match end string	<input type="checkbox"/>	*M	<input checked="" type="checkbox"/>	
stripLevel	stripLevel	Strip filter#	<input type="checkbox"/>	*M	<input checked="" type="checkbox"/>	
title	title	Result title	<input type="checkbox"/>	*M	<input checked="" type="checkbox"/>	

The panel contains the following parameters:

Parameter	Value
url	The URL of the source page. If you do not include the protocol in the URL, Portal-to-Go prepends this value with "http://".
beginTag	A string that matches the start point of the text retrieved from the URL target page. This value is optional. If not specified, the start point is the beginning of the page.
endTag	A string that matches the end point of the text retrieved from the URL target page. This value is optional. If not specified, the end point is the end of the page.
stripLevel	<p>The strip level identifies the class used by the stripper adapter to filter the retrieved content. Portal-to-Go provides two levels:</p> <ul style="list-style-type: none"> ■ 0: Retains all markup tags in the content. ■ 1: Removes all markup tags in the content. <p>You can implement additional strip levels or filters. For more information, see "Extending the Stripper Adapter" in Chapter 9, "Adapters".</p>
title	An optional title of the result.

URL Adapter Input Parameters This adapter allows you to retrieve XML documents that are saved on the hard drive of your local machine and that conform to the SimpleResult DTD every time you want to bring up the service. You can also store static pages from the Web and retrieve them with this adapter from the hard drive of your local machine. For a service that is based on the URL adapter, the Input Parameters panel appears as follows:

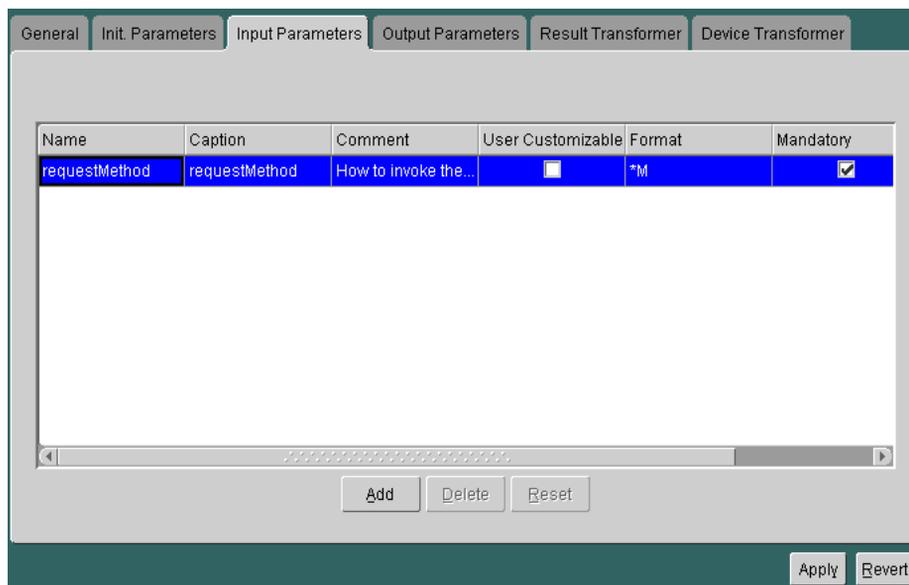
Name	Caption	Comment	User Customizable	Format	Mandatory
URL	URL	Data Source URL	<input type="checkbox"/>	*M	<input checked="" type="checkbox"/>

Buttons: Add, Delete, Reset, Apply, Revert

The panel contains the following parameter:

Parameter	Value
URL	<p>The URL of the XML document that is retrieved by the URL Adapter. The XML document should conform to the SimpleResult DTD.</p> <p>For example, if the URL parameter is <code>file:///d:/SimpleResultExample1.xml</code>, then the adapter gets the file <code>d:/SimpleResultExample1.xml</code> from the hard drive on the local machine. If the URL parameter is <code>http://ptg1.oracle.com/SimpleResultExample2.xml</code>, then the adapter gets the file <code>SimpleResultExample2.xml</code> from the host <code>ptg1.oracle.com</code>.</p>

Servlet Adapter Input Parameters For a service that is based on the Servlet adapter, the Input Parameters panel appears as follows:



Parameter	Value
requestMethod	<p>The value of this parameter can be any of the following: GET, HEAD, POST, PUT, DELETE, OPTIONS, or TRACE.</p> <p>Depending on the value of this parameter, the appropriate method of the servlet gets called when the adapter is invoked.</p> <p>For example, if the value of requestMethod is GET, then the doGet method of the servlet is called.</p>

You can also define other input parameters and add them by clicking on the Add button. For more information, see "[Creating a Service Using the Servlet Adapter](#)".

Output Parameters Panel

The output parameters panel allows you to set captions for service output parameters.

Result Transformer Panel

You use the Result Transformer panel to specify a transformer for Portal-to-Go to use to convert Adapter Result content. Portal-to-Go provides two content formats, Adapter Result and Simple Result. Adapter Result is intended to be an intermediary format for passing raw data between services. Device transformers, which convert service content for the target format, cannot convert Adapter Result format. A result transformer must therefore convert the content to Simple Format before it can be processed by a device transformer.

To create a result transformer:

1. Click Add.
2. Enter the name of the result transformer in the name field. You must use the same name for the result transformer as you used for the WIDL service to which it applies. This is the value of the `PAsection` input parameter.
3. Click Edit. The XSL Editor window appears. The XSL Editor is a simple text editor you can use to build your result transformer. You can cut and paste from other environments into the XSL Editor.
4. When finished, click OK.

Device Transformer Panel

The Device Transformer panel lists the logical devices in the repository. You can specify a custom transformer to be used with the master service for a logical device. A custom transformer enables you to optimize the presentation of service content for a particular device. Since the transformer is specialized for a particular device and master service, you can associate a custom transformer with only one master service and one logical device.

Deleting a Master Service

You can delete a master service in the Service Designer as follows:

1. Select the master service that you want to delete.
2. Right-click and select Delete.
3. Confirm the action.

When you delete a master service, the Service Designer flags any aliases to the master service.

Creating a Service Using the Servlet Adapter

The Servlet adapter enables you to integrate other applications you might have which are already Java servlets. This provides a convenient way to call them as Portal-to-Go services and make them wirelessly available.

To create a Portal-to-Go service using the Servlet adapter:

1. Create a directory in the classpath oracle.panama.adapter.servlet for the **.class** file. You should not experience difficulties in creating The MasterService if you also have the **.class** file in a similar directory where you are running the Service Designer.
2. Create a regular master service with the ServletAdapter.
3. Specify oracle.panama.adapter.servlet.TestServlet as the class name.
4. Specify RequestMethod as GET.
5. Add input parameters "param1" and "param2" to the master service.
6. Create an alias to this service for a service hierarchy.

Anything else you want to add is an exercise in building Java adapters.

Example:

Here is a Java file and a "make" file for a simple example.

```

-----
set JAVA_HOME=D:\Progra~1\jdk122
set
CLASSPATH=.;%JAVA_
HOME%\jre\lib\rt.jar;d:\ptg102\Server\panama\server\classes;d:\ptg102\Server\pan
ama\server\papz;d:\ptg102\Server\panama\lib\panama_
core.zip;d:\ptg102\Server\panama\lib\server.zip;d:\ptg102\Server\panama\lib\clie
nt.zip;d:\ptg102\Server\panama\lib\xmlparserv2.jar;d:\ptg102\Server\panama\lib\c
lasses111.zip;d:\ptg102\Server\panama\lib\jndi.jar;d:\ptg102\Server\panama\lib\s
ervlet.jar;.

%JAVA_HOME%\bin\javac TestServlet.java

-----

package oracle.panama.adapter.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import oracle.panama.core.admin.L;

public class TestServlet extends HttpServlet {

    public void init(ServletConfig cfg) throws ServletException {
        L.n ("servlet initialized");
    }

    public void doDelete (HttpServletRequest input, HttpServletResponse
output) throws IOException, ServletException {
        L.n ("doDelete() invoked");
    }

    public void doGet (HttpServletRequest input, HttpServletResponse output)
throws IOException, ServletException {
        L.n ("doGet() invoked");
        PrintWriter p = output.getWriter();
        p.println ("<?xml version=\"1.0\">");
        p.println ("<SimpleResult>");
        p.println ("<SimpleText title=\"ServletAdapter Test\">");
    }
}

```

```

        p.println("<SimpleTextItem title=\"Static text\">Hello
World</SimpleTextItem>");

        String p1 = input.getParameter ("param1");
        String p2 = input.getParameter ("param2");

        p.println("<SimpleTextItem title=\"Parameters\">param1 = \"\" + p1 +
\"\" and param2 = \"\" + p2 + \"\"</SimpleTextItem>");
        p.println("</SimpleText>");
        p.println("</SimpleResult>");

    }

    public void doOptions (HttpServletRequest input, HttpServletResponse
output) throws IOException, ServletException {
        L.n ("doOptions() invoked");
    }

    public void doPost (HttpServletRequest input, HttpServletResponse
output) throws IOException, ServletException {
        L.n ("doPost() invoked");
    }

    public void doPut (HttpServletRequest input, HttpServletResponse output)
throws IOException, ServletException {
        L.n ("doPut() invoked");
    }

    public void doTrace (HttpServletRequest input, HttpServletResponse
output) throws IOException, ServletException {
        L.n ("dotrace() invoked");
    }

}

```

Service Trees

A service tree is a folder you use to organize and distribute access to services. It can contain master services, service aliases, or other folders.

Folders make services accessible to users. When you create a new user, you also create a service tree for that user. Any service that you place in a user's service tree (or user's home), is accessible to the user.

Similarly, when you create a group, you specify the service trees that belong to the group. Any member of the group can access the services in that group's service trees. While a user can have only one private service tree, groups can have many service trees. Different groups can share the same service tree.

If you do not specify an owner when you create a service tree, the tree is public. If you specify an owner, the service tree is private. End users can access any public service tree, but they can only access their own private service trees. They have complete control of their service trees from the Personalization Portal; they can copy services between service trees, rename services, and create folders.

Portal-to-Go does not provide dependency tracking for a private service (a service in a private service tree). If the target service of the private service is no longer valid, an error occurs when a user attempts to submit a request for that service. Portal-to-Go then advises the user to delete the service from the private service tree.

Creating a Folder

You can create folders in either the Master Services folder or the Service Trees folder of the Portal-to-Go repository tree. To create a folder, you use the Create New Folder form.

To invoke the form:

1. Select the root folder under which you want to place the folder.
2. Right click.
3. Select Create New Folder.

The Create New Folder form appears as follows:

The panel includes the following parameters:

Parameter	Value
Name	The folder name must be a unique name in the parent folder.
Valid	Select the Valid check box to enable the folder.
Visible	Select the Visible check box to make the folder accessible to users.
Owner	The user or group that owns the folder. Only the owner can modify the folder.
Sequence	<p>This integer value lets you alter the order in which services and folders appear on output devices. By default, these appear in order by sequence number, then name. You can enter values in the sequence fields to rearrange the order in which the services and folder appear.</p> <p>By default, Portal-to-Go sorts services and folders in ascending order by sequence number, then by name. You can change this behavior by altering the <code>order.services</code> property in the System.properties file. You can specify sorting by sequence number, name, or date last written, in either ascending or descending order.</p>
Area Id	An integer value that allows you to specify whether a folder is visible based on a user's physical location. This enables you to implement location-specific services. This is an optional value.

Parameter	Value
Description	An optional description of the folder.

Creating a Service Alias

A service alias is a link to a master service, folder, or other alias. Service aliases help you to distribute service access to multiple users or groups. They also enable you to specialize master services, since default parameter values specified by an alias override values set at the master service. This characteristic provides several benefits. One benefit is that it enables you to localize services. For example, suppose you create a service that delivers restaurant information for a city. The adapter takes a single parameter, a location, and returns a list of restaurants in the area. While the master service can specify a more general location, such as the city, you can create aliases that provide a more specific parameter, such as a district within the city. You can then distribute the aliases, as appropriate, to user groups that you assemble based on the users' locations of residence.

To create a service alias:

1. Highlight and right-click any Service Tree subfolder in the Portal-to-Go Repository tree.
2. Click Create New Alias.

The first form in the sequence appears as follows:

The panel includes the following parameters:

Parameter	Value
Name	The service alias name. This must be a unique name within the parent folder.
Service	The master service referenced by the service alias. To select a master service, click Browse and choose a master service from the Browse Services window.
Valid	Select the Valid check box to enable the alias.
Visible	Select the Visible check box to make the alias accessible to users.
Owner	The user or group that owns the alias. Only the owner can modify the alias. If you are creating an alias in a user's home folder, you can make the owner that user. This enables the user to modify the service—by renaming it, for example—at the personalization portal.

Parameter	Value
Sequence	<p>This integer value lets you alter the order in which services and folders appear on output devices. By default, these appear in order by sequence number, then name. You can enter values in the sequence fields to rearrange the order in which the services and folders appear.</p> <p>By default, Portal-to-Go sorts services and folders in ascending order by sequence number, then by name. You can change this behavior by altering the <code>order.services</code> property in the System.properties file. You can specify sorting by sequence number, name, or date last written, in either ascending or descending order.</p>
Description	An optional description of the alias.

Complete the properties in the form and click Finish to create the alias in the repository. You do not need to complete the second form in the sequence. You can configure the runtime parameters for the alias by modifying the alias properties, just as you would for the master service on which the alias is based. For more information on specifying runtime parameters, see "[Input Parameters Panel](#)".

Creating a Bookmark

A bookmark is a Portal-to-Go service that points to an external resource. The external resource is typically a Web page that serves content in a format supported by the target device.

Portal-to-Go does not process the content of the bookmark target. As a result, bookmark services are not available to all targeted device, as are other Portal-to-Go services. In most cases, bookmarks are set in the personalization portal by the end user, not in the Service Designer.

To create a bookmark in the Service Designer, you use the Create New Bookmark form. To invoke the form:

1. Highlight and right-click Master Services, or any of its subfolders, in the Portal-to-Go repository tree.
2. Click Create New Bookmark.

The Create New Bookmark form appears:

The panel includes the following parameters:

Parameter	Value
Name	The bookmark name. This must be a unique name within the parent folder.
Valid	Select the Valid check box to enable the bookmark.
Visible	Select the Visible check box to make the bookmark accessible to users.
Owner	The user or group that owns the bookmark. Only an owner can modify the bookmark.
HREF	The URL target.
Sequence	This integer value lets you alter the order in which services and folders appear on output devices. By default, these appear in order by sequence number, then name. You can enter values in the sequence fields to rearrange the order in which the services and folder appear. By default, Portal-to-Go sorts services and folders in ascending order by sequence number, then by name. You can change this behavior by altering the <code>order.services</code> property in the System.properties file. You can specify sorting by sequence number, name, or date last written, in either ascending or descending order.
Description	An optional description for the bookmark.

Managing Users and Groups

This document describes how you can create and modify Portal-to-Go users and user groups. Topics include:

- [User Roles](#)
- [Portal-to-Go Users](#)
- [User Groups](#)
- [Integrating Users with Existing Provisioning Systems](#)

User Roles

Portal-to-Go users can be granted the following roles:

- Administrator
- Designer
- Anonymous

Users assigned to the Administrator role can change the Portal-to-Go configuration and can start and stop any component, even the entire server. In addition, users with the Administrator role can update privileges on any part of the Portal-to-Go repository and can install and remove any service.

Users assigned the Designer role can create and modify any service or adapter.

Users having the Anonymous role are Portal-to-Go end users who have access only to the objects they own.

Portal-to-Go Users

This section describes how you use the Portal-to-Go Service Designer to accomplish the following user-related tasks:

- [Creating Users](#)
- [Adding Agent Information to a User](#)
- [Deleting Users](#)
- [Modifying Users](#)

Creating Users

You use the Create New User form to create users in the Portal-to-Go repository. To invoke the form:

1. Click Users in the Portal-to-Go repository tree view. The Query Users form displays in the right panel.
2. Click Add to display the Create New User form.

Create New User

Name:

External ID:

Display Name:

Password:

Password Hint:

User Root Parent:

(A new folder will be created in the selected folder)

Group:

Enabled

Administrator Designer Anonymous

The form includes the following parameters:

Parameter	Value
Name	A unique name for the user.
External ID	The user's unique identifier in an external provisioning system, such as a telephone or an account number.
Display Name	The name displayed in the service tree.
Password	The user's password.
Password Hint	A hint that helps the user remember a forgotten password.
User Root Parent	The root folder in which the Service Designer creates the user's folder.
Group	The group to which the user belongs. Any service that is available to the selected group will be available to the new user.
Enabled	Select this check box to make the user valid.
Administrator	Select this check box to grant the user administrator privileges.
Designer	Select this check box to grant the user designer privileges.
Anonymous	Select this check box to grant the user access to only public Portal-to-Go services.

Adding Agent Information to a User

Once Portal-to-Go administrators create new users, they can add agents to user properties. Agents associate a user with a channel for delivering asynchronous requests. For example, when a user schedules an asynchronous request, its result can be sent to either an SMS-enabled phone or to an email inbox. Each agent uses a logical device definition to transform the result of the user request for output to the appropriate channel. An agent has an address, which is a protocol-specific parameter required to deliver the result to a channel of the user's choice, such as an email address. Adding agent information to a user's properties enables a user to receive notifications.

To add agent information for a user:

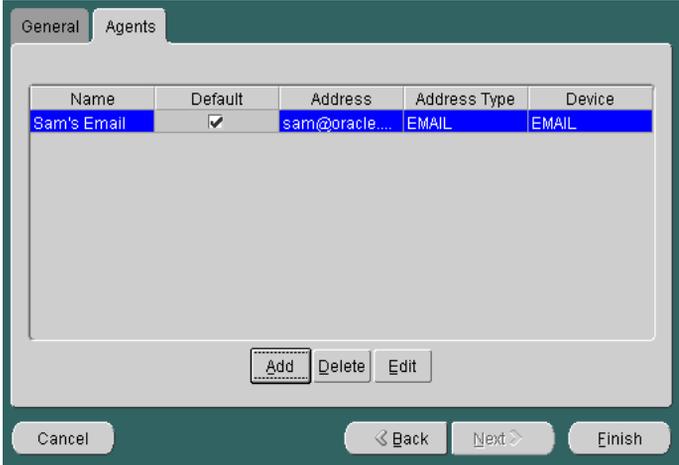
1. Select Users in the Portal-to-Go repository tree. The Query Users form appears.

The screenshot shows a 'General' tab interface. At the top left is a tab labeled 'General'. Below it are two search rows. The first row is labeled 'Name' and contains a dropdown menu with 'contains' selected, a text input field, and a 'Find' button. The second row is labeled 'Disp. Name' and contains a dropdown menu with 'contains' selected, a text input field, and a 'Find' button. Below these is a table with four columns: 'Name', 'Display Name', 'External ID', and 'Root Folder'. The table body is currently empty. At the bottom of the form are three buttons: 'Add', 'Delete', and 'Properties'.

The form includes the following fields:

Field	Enter
Name	A string for querying the user's name.
Disp. Name	A string for querying the user's display name.

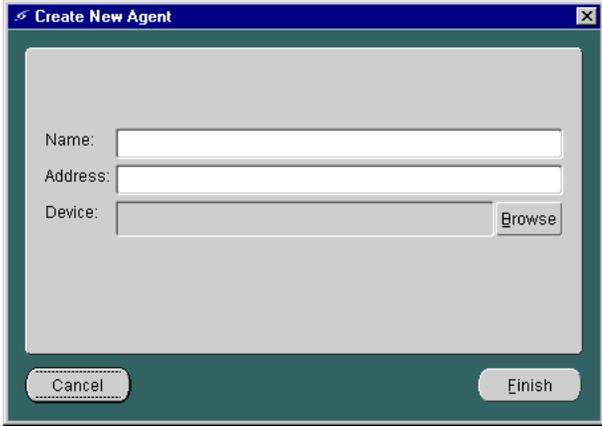
2. Enter values in the Name or Disp. Name field and click Find. The query form returns a list of users that match your search criteria.
3. Select the user to which you want to add agent information.
4. Click Properties to display the Properties form for that user.
5. In the Properties form, click the Agents tab to display the Agents panel. The panel is blank if you have just created a new user.



The Agents tab includes the following attributes:

Attribute	Value
Name	A unique name for the user agent. This name appears at the Personalization Portal.
Default	When selected, makes that agent the user's default agent. Only one agent can be selected as default.
Address	Address of the user's device. This can be an email address for notifications, or a phone number for SMS (Short Message Service) notifications.
Address Type	The Service Designer generates this parameter's value based on the value of the Device parameter.
Device	The name of the logical device.

6. Click Add to display the Create New Agents form.



The Create New Agents form includes the following parameters:

Parameter	Description
Name	A unique name for the user agent. This name appears at the Personalization Portal.
Address	Address of the user's device. This can be an email address for notifications, or a phone number for SMS (Short Message Service) notifications.
Device	The name of the logical device. Click Browse to display a list of available devices.

7. Provide values for the new agent parameters and click Finish to complete the user agent configuration.

Deleting Users

Follow these steps to delete a user:

1. Select Users in the Portal-to-Go repository tree view to display the Users form.
2. Enter the name of the user you wish to delete in the Name field or enter the user's identifier in the External ID field.
3. Click Find to display users that match your search criteria. To display all users, leave the Name and External ID fields blank and then click Find.

4. Select the user you want to delete.
5. Click Delete.
6. Confirm the action by clicking the Yes button in the Confirm Delete dialog.

Modifying Users

Follow these steps to modify a user's information:

1. Select Users in the Portal-to-Go repository tree view to display the Users form.
2. Enter the name of the user you wish to delete in the Name field or enter the user's identifier in the External ID field.
3. Click Find to display users that match your search criteria. To display all users, leave the Name and External ID fields blank and then click Find.
4. Select the user whose information you want to modify.
5. Click Properties to display the Properties form for this user.
6. Modify the desired parameters on the General and Agents tabs as described in "[Adding Agent Information to a User](#)", and click the Apply button to save your changes.

User Groups

This section describes how you can create, modify, or delete user groups using the Portal-to-Go Service Designer:

- [Creating User Groups](#)
- [Deleting Groups](#)
- [Adding Users to a Group](#)
- [Adding Services to a Group](#)
- [Removing Users from a Group](#)
- [Removing Services from a Group](#)
- [Changing a User's Group](#)

Creating User Groups

Portal-to-Go users can be organized into groups, with each Portal-to-Go user belonging to a single group. Services can also be assigned to these user groups and can be made available to all users in that group. To create a group:

1. Right-click Groups in the Portal-to-Go repository tree view.
2. Click Create New Group to display the Create New Group form.

The Create New Group form includes the following parameters:

Parameter	Value
Name	A unique name for the group.
Available Users: Name	A string for querying the user's name from the list of available users.
Available Users: Disp. Name	A string for querying the user's display name from the list of available users.
Group Members: Name	A string for querying the user's name from the group members.

Parameter	Value
Group Members: Disp. Name	A string for querying the user's display name from the list of selected users.

3. Enter the name of the new user group in the Name field at the top of the screen.
4. In the Available Users section of the Create New Group form, enter the name of the user you wish to add in the Name field or enter the user's display name.
5. Click Find. To display all available users, leave the Name and Disp. Name fields blank and click Find. Results of the search criteria are displayed by user name and display name in the Available Users section.

Note: Searching the Name and Disp. Name fields in the Available Users section returns only users who are not assigned to any group.

6. Select the user you want to add to the group. You can select multiple users by holding the CONTROL key.
7. Click the right arrow (>) to move users from the Available Users section to the Group Members section. Clicking the double right arrows (>>) adds all available users to the new group.
8. Click Next.

After creating the group, you can make services available to the group members by assigning service folders to the group. You can only assign services by folder; you cannot assign individual services to a group. To assign services to a group:

1. If not already selected, select the group from the Portal-to-Go tree view.
2. Click the Service Folders tab.
3. Expand, if necessary, the root service folders shown to display the folder you want to make available to the group.
4. Select the folder you want to assign to the group. Holding the CONTROL key enables you to select multiple folders.
5. Click the right arrow (>) to move folders to the Selected Service Folders field. Clicking the double right arrows (>>) moves all folders to the Selected Services Folder field.

6. Click Finish.

Deleting Groups

To delete a group:

1. Expand the Groups folder in the Portal-to-Go repository tree view.
2. Select the group you want to delete by clicking the right mouse button.
3. Click Delete.
4. Click Yes in the Confirm Delete dialog.

Adding Users to a Group

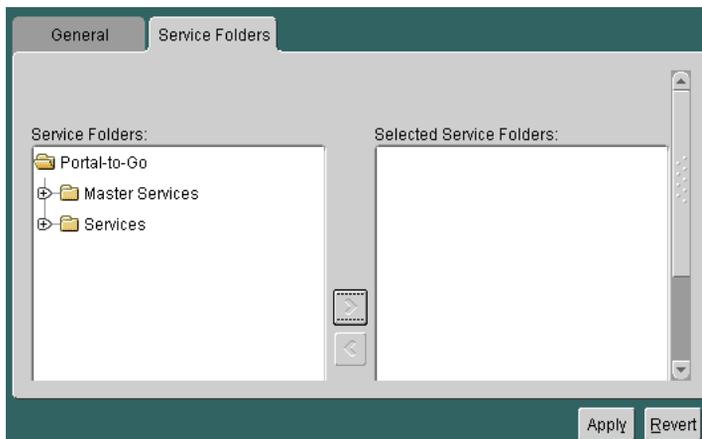
To add a user to a group:

1. Expand the Groups folder in the Portal-to-Go repository tree view.
2. Select the group to which you want to add users. The Group Properties form appears in the right frame. Select the General tab, if it does not already appear.
3. In the Available Users section, enter the name of the user you wish to add in the Name field or enter the user's display name.
4. Click Find. To display all available users, leave the Name and Disp. Name fields blank and click Find. Results of the search criteria are displayed by user name and display name in the Available Users section.
5. Select the user you want to add to the group. You can select multiple users by holding the CONTROL key.
6. Click the right arrow (>) to move users from the Available Users section to the Group Members section. Clicking the double right arrows (>>) adds all available users to the group.
7. Click Apply.

Adding Services to a Group

To make services available to a group:

1. Expand the Groups folder in the Portal-to-Go repository tree view.
2. Select the group name to which you want to add services.
3. Click the Service Folders tab to display the Service Folders panel.



4. Under Service Folders, expand the folders, if necessary, to display the folder that you want to assign to the group. Select the folder. You can select multiple folders by holding the CONTROL key.
5. Click the right arrow (>) button to move the folder to the Selected Service Folders field.
6. Click Apply.

Removing Users from a Group

To remove users from a group:

1. Expand the Groups folder in the Portal-to-Go repository tree view to display the available groups.
2. Select the group from which you want to remove users. The group properties form appears in the right frame. Select the General tab, if it does not already appear.
3. In the Available Users section, enter the name of the user you wish to remove from the group in the Name field or enter the user's display name.
4. Click Find. To display all available users, leave the Name and Disp. Name fields blank and click Find. Results of the search criteria are displayed by user name and display name in the Available Users section.

5. Select the user you want to remove from the group. Hold the CONTROL key to select multiple users for removal.
6. Click the left arrow (<) to move users from the Group Members section to the Available Users section. Clicking the double left arrows (<<) removes all users from the group.
7. Click Apply.

Removing Services from a Group

To remove services from a group:

1. Expand the Groups folder in the Portal-to-Go repository tree view to display the available groups.
2. Select the group from which you want to remove services. The group properties form appears in the right frame.
3. Click the Folders tab.
4. Under Selected Service Folders, select the folder you want to remove. You can select multiple folders by holding the CONTROL key.
5. Click the left arrow (<) to remove the folder from the Selected Service Folders field. Clicking the double left arrows (<<) removes all folders from the group.
6. Click Apply.

Changing a User's Group

A Portal-to-Go user can only belong to one group at a time. If you wish to change a user's group, you must first remove the user from one group before adding the user to another group. See ["Removing Users from a Group"](#) and ["Adding Users to a Group"](#) for more information.

Integrating Users with Existing Provisioning Systems

You can integrate your repository with an existing provisioning system using one of the following:

1. You can manage users with the Portal-to-Go provisioning adapter. The provisioning adapter supports Portal-to-Go services that create, update, and delete users at runtime.

2. Use the `LoadXml` utility to import users from a flat XML file that conforms to the Repository DTD. This utility enables you to create and update user information. For more information on the `LoadXML` utility, see [Chapter 3, "Portal-to-Go Tools"](#).
3. You can look up users at runtime by using the `UserAuthenticationHook` and the `ProvisioningHook` interfaces. This is similar to using the `LoadXML` utility, but with the hook interfaces you can import users one-by-one. The `LoadXML` utility operates on the entire repository. The following sections describe these interfaces.

ProvisioningHook Interface

You use the `ProvisioningHook` interface to integrate an external provisioning system with Portal-to-Go. Portal-to-Go locates the provisioning class to use in the **System.properties** file as the value of the `locator.provisioning.class` parameter. The implementing class must be implemented as a singleton class. This interface has the following methods:

- `getRootFolder()`: Creates a root folder for a user.
- `getUsersRootFolder()`: Gets the common root folder for all users.

UserAuthenticationHook Interface

Portal-to-Go calls the user authentication class when it cannot locate a user. The class must be implemented as a singleton class. This interface has the following methods:

- `authenticate()`: Authenticates a Portal-to-Go user.
- `checkUser()`: Checks if the user exists, and if it does, sets appropriate attributes.

Rebranding the Personalization Portal

This document explains the Personalization Portal architecture. Topics include:

- [Overview](#)
- [Use Sequence](#)
- [Directory Structure](#)
- [Login Sequence](#)
- [Main Page Components](#)
- [Customization Components](#)
- [Customizing Services](#)
- [Customizing Folders](#)
- [Configuring Jobs](#)
- [Configuring the User Profile](#)
- [Debug Messages](#)
- [National Language Support](#)

Overview

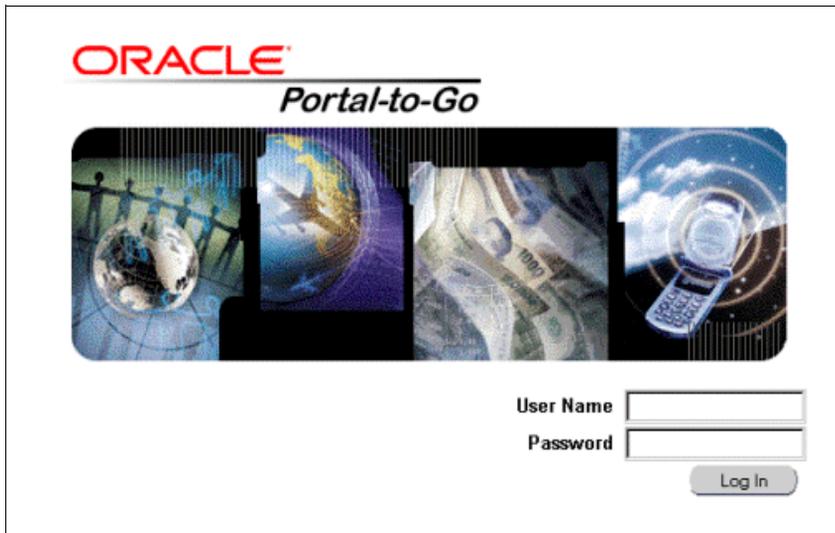
The Personalization Portal is a Web-based interface that allows end-users to manage their mobile device portals. At the Personalization Portal, users select the services that appear on their mobile devices, store frequently used parameters (such as email addresses and account numbers), and create notifications. Any change that a user makes at the Personalization Portal is immediately reflected on the users mobile device.

The Portal-to-Go Personalization Portal is both a framework for the personalization interface and a sample implementation of that framework. The framework consists of JavaServer Pages (JSP) files, JavaBean modules, JavaScript, and such static elements as images, style sheets, and HTML files. Another element of the framework is the logical sequence in which the elements execute. You can rebrand the Personalization Portal based on the existing framework or, by altering the logic in the JSP files and JavaBeans, restructure the framework itself.

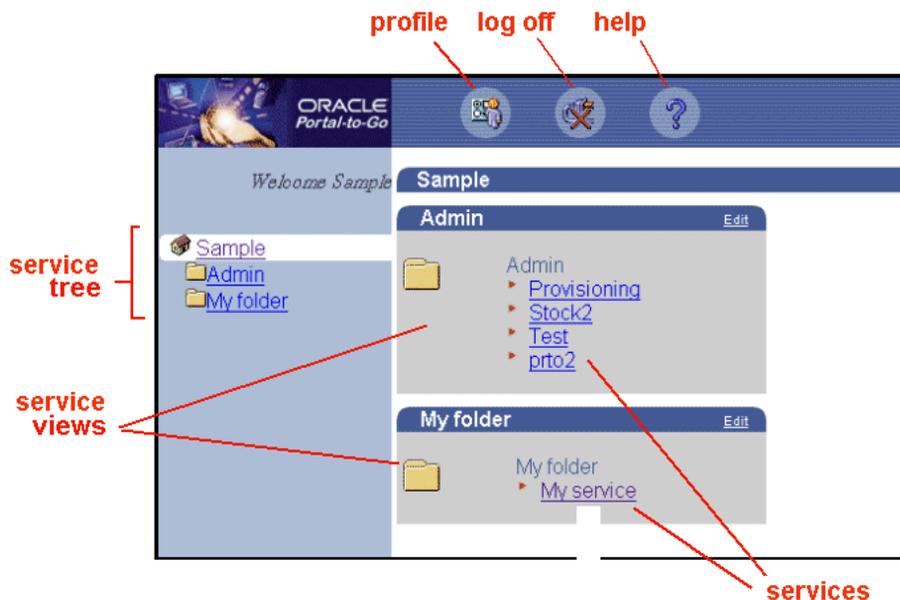
The following sections acquaint you with the elements that generate the portal and their order of execution, as well as the file naming conventions and the directory structure used.

Use Sequence

A user accesses the Personalization Portal by entering a valid user name and password at the login page. The login page is generated by the JSP file, **login.jsp**.

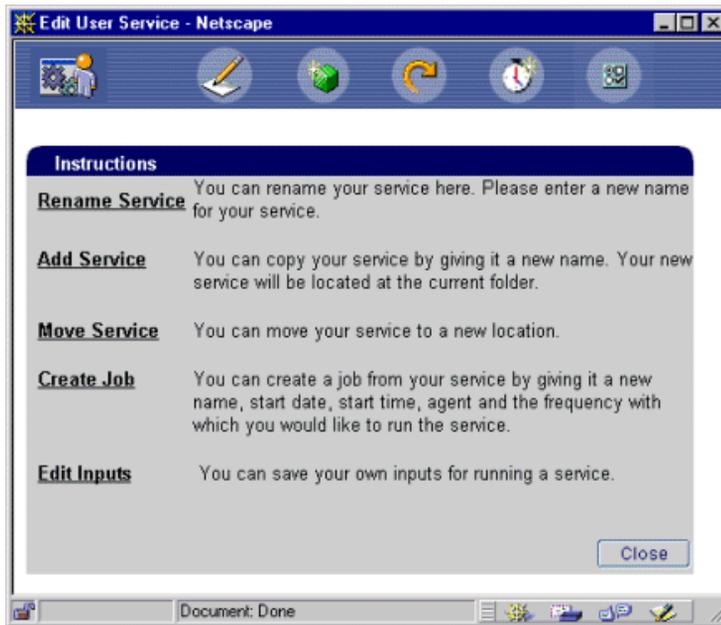


If the user name and password is valid, the user's main page appears. The JSP file **PapzMain.jsp** generates the main page, which appears as follows:

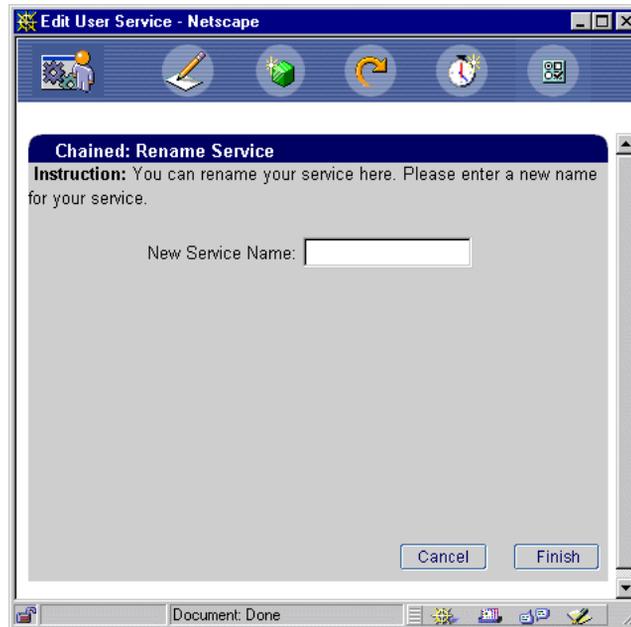


The service tree provides an easy way for users to browse through their folders and services. The service view presents a desktop view of the user's folders and services. The service view includes links for invoking and editing services. The edit link allows users to customize their services. Users can add, move, and rename folders and services. For services, users can also create jobs (or notifications) and store service input values. For folders, users can also enable or disable services in that folder and create bookmarks.

When a user clicks the edit link for a service, for example, the following window appears:



When a user clicks a link in the Edit User Service window, a form appears where the user completes the action. For example, when a user clicks the Rename Service link, the following form appears:



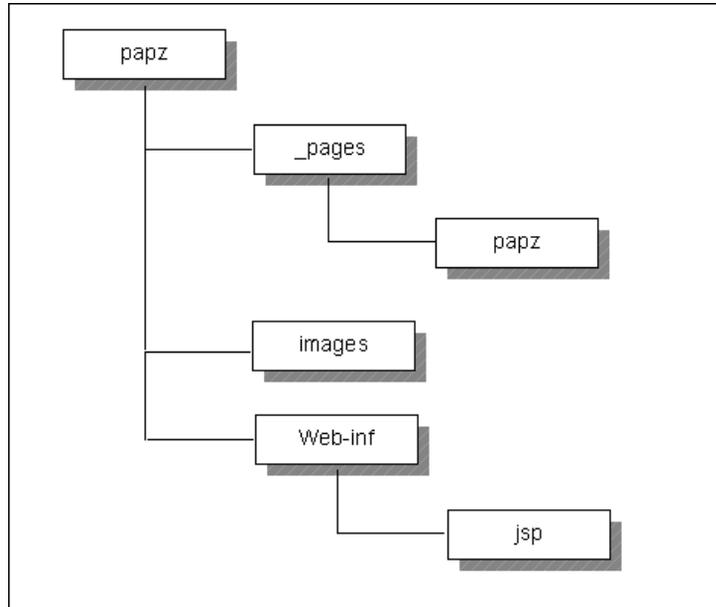
The form contains instructions for renaming a service and a text field where the user enters a new value for the service name.

Note: The user must be the owner of the service to be able to change the service. You grant ownership privileges in the Service Designer. For a user, you would typically grant ownership to a service alias rather than the master service. For more information, see ["Creating a Service Alias"](#) in [Chapter 5, "Portal-to-Go Services"](#).

When finished, users click the Log Off button to close the Personalization Portal session.

Directory Structure

To rebrand the Personalization Portal, you modify the files that generate the Personalization Portal. After installing Portal-to-Go, these files are located in the papz directory. The papz directory has the following structure:



The directory contain the following:

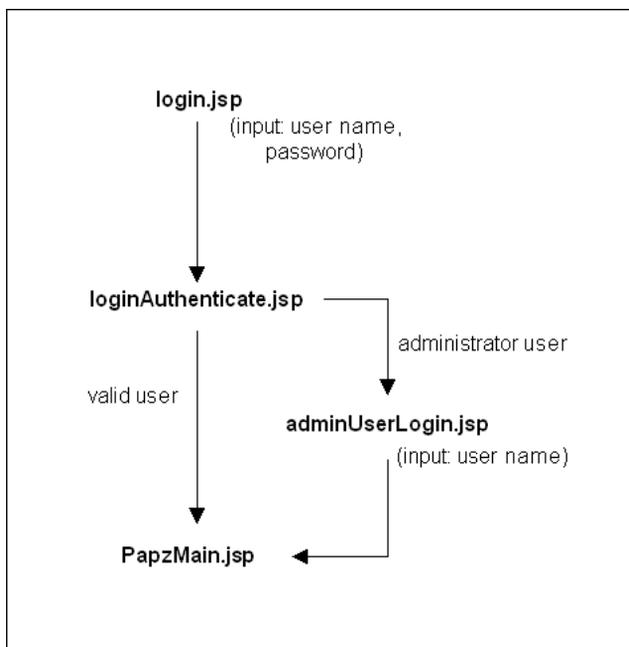
Directory	Contents
papz	Container JSP files. Container files are accessed directly by browsers. They invoke Java beans and other JSP files. This directory also contains style sheets and JavaScript files.
papz/_pages/papz	The folder where the compiled pages are saved.
papz/images	Images used throughout the Personalization Portal.
papz/Web-inf/jsp	Module JSP files. These files are included by either container JSP files or other module JSP files. The edit and action JSP files are in this directory.
papz/oracle/panama/person	JavaBean modules.

Login Sequence

The **login.jsp** file generates the user login page. When the user clicks the Log In button, the flow moves to the **loginAuthenticate.jsp** file. The **loginAuthenticate.jsp** file validates the user. There are three possible results:

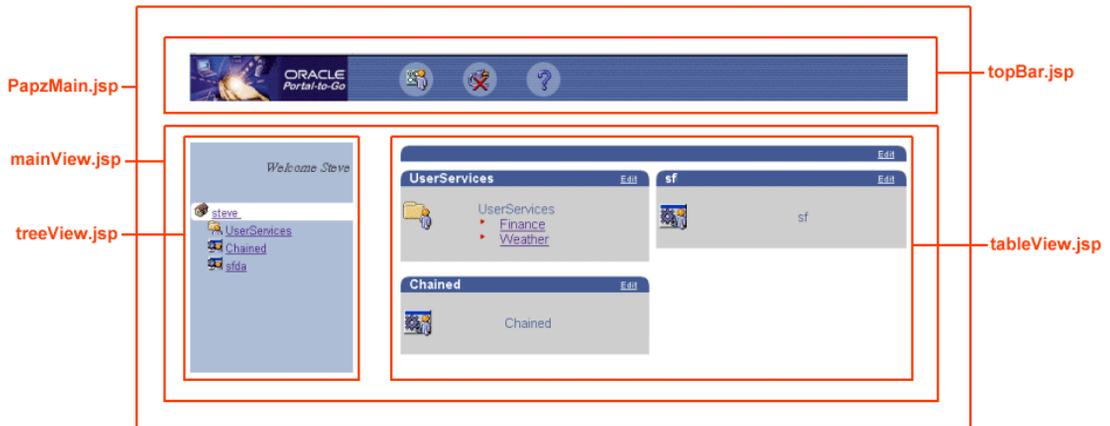
- If the user name and password are correct, the flow moves to the **PapzMain.jsp** file.
- If the user name or password are incorrect, an error message is displayed and the flow moves back to the **login.jsp** file.
- If the user is an administrator, the flow moves to the **adminUserLogin.jsp** file. This file generates an input form where the administrator can enter the user name of the end user whose account the administrator wants to manage. If the user name is correct, the flow moves to **PapzMain.jsp** (user home). If the user name is incorrect, the flow moves back to the **login.jsp** file.

The flow of a login action is displayed in the following figure.



Main Page Components

The JSP file **PapzMain.jsp** generates the Personalization Portal home page. **PapzMain.jsp** is a container file; it is accessed directly by browsers. It invokes other JSP files to generate each component of the main page. The following illustrates the files that generate the Personalization Portal home page.



topBar.jsp generates the function buttons at the top of the page. These buttons enable end users to change their profiles, log off, and invoke help. **treeView.jsp**, which is included by **mainView.jsp**, presents a tree view of the services available to the user. **tableView.jsp** provides a desktop view of folders and services, and includes links for editing service and folder attributes.

Customization Components

The edit and action JSP files execute the tasks associated with user customization:

- The edit JSP files, which begin with "ed", generate the forms for accepting user inputs.
- The action JSP files, which begin with "do", process user inputs, perform the action, and display the result.

For example, to rename a folder, Portal-to-Go first invokes **edFolderRenameForm.jsp**, then **doFolderRename.jsp**. The edit files are invoked by the JavaScript file **papz.js**.

Users can customize or configure the following:

- Services
- Folders
- Jobs (jobs, or notifications, alert users when preset conditions are met)
- Profile information
- Bookmarks

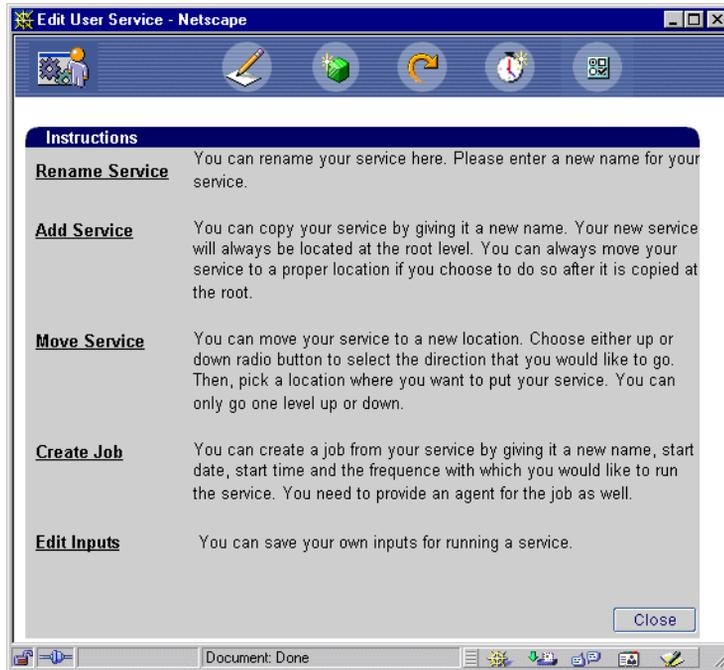
The following sections discuss the files that execute the customization and the sequences in which they execute.

Customizing Services

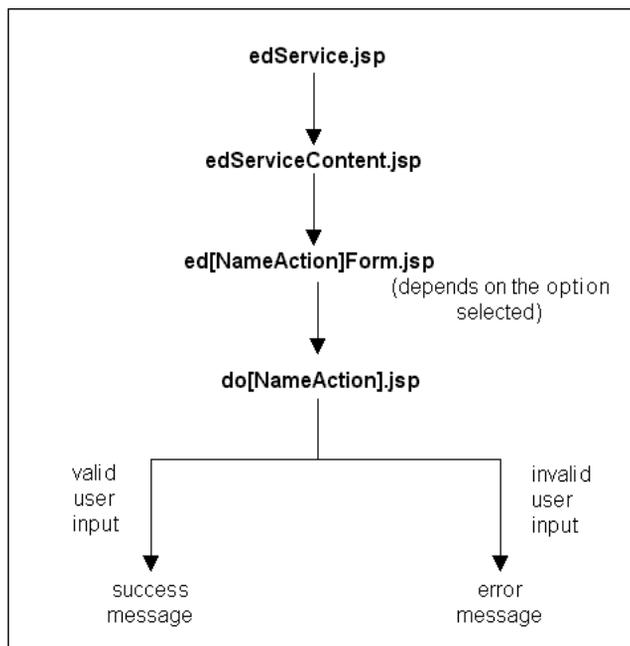
The **edService.jsp** file displays the Edit Service dialog, which allows users to customize their services. The dialog contains two frames:

- The top frame (**edServiceMenu.jsp**) displays the menu options available to the user.
- The bottom frame (**insService.jsp**) displays the instructions for all the menu options.

The following figure shows the edit service window.



Depending on the option selected by the user, an input form is displayed in the bottom frame by an `edNameActionForm.jsp` file. When the user clicks the Finish button on the input form, the flow moves to a `doNameAction.jsp` file. The `doNameAction.jsp` file posts and processes input values, and, depending on the validity of the input values, displays a success or an error message. The following figure displays the flow of control when a user edits a service.



The following lists the actions a user can perform in the Edit Services dialog and the corresponding JSP files.

Action	Description	Action JSP File	Do Action JSP File
Rename service	Rename a service	edServiceRenameForm.jsp	doServiceRename.jsp
Add service	Add a service by giving it a new name. The new service is always located at the current folder.	edServiceCreateForm.jsp	doServiceCreate.jsp
Move service	Move a service to a new folder.	edServiceMoveForm.jsp	doServiceMove.jsp

Action	Description	Action JSP File	Do Action JSP File
Create job	Create a job for a service by giving it a name, start date, start time, frequency, and an agent.	edJobForm.jsp	doJobCreate.jsp
Edit inputs	Save a set of values for running a service.	edServiceParamForm.jsp	doServiceSaveParam.jsp

Customizing Folders

The architecture of the edit folder files is similar to that of the edit services files. The **edFolder.jsp** frameset file invokes the Edit Folder dialog, which allows users to customize their folders. The dialog contains two frames:

- The top frame (**edFolderMenu.jsp**) displays the menu options available to the user.
- The bottom frame (**insFolder.jsp**) displays the instructions for all the menu options.

Depending on the option that the user selects in the top frame, the flow moves to a **edNameAction.jsp** file. This file displays an input form in the bottom frame. Once the user clicks the Finish button on the input form, the flow moves to **doNameAction.jsp** file. The **doNameAction.jsp** file posts and processes the input values, and, depending on the validity of the input values, displays a success or an error message.

The following lists the actions a user can perform in the Edit Services dialog and the corresponding JSP files.

Action	Description	Action JSP File	Do Action JSP File
Rename folder	Rename a folder.	edFolderRenameForm.jsp	doFolderRename.jsp
Add folder	Add a folder under the current folder.	edFolderAddForm.jsp	doFolderAdd.jsp
Move folder	Move a folder to any other folder under the root location.	edFolderMoveForm.jsp	doFolderMove.jsp

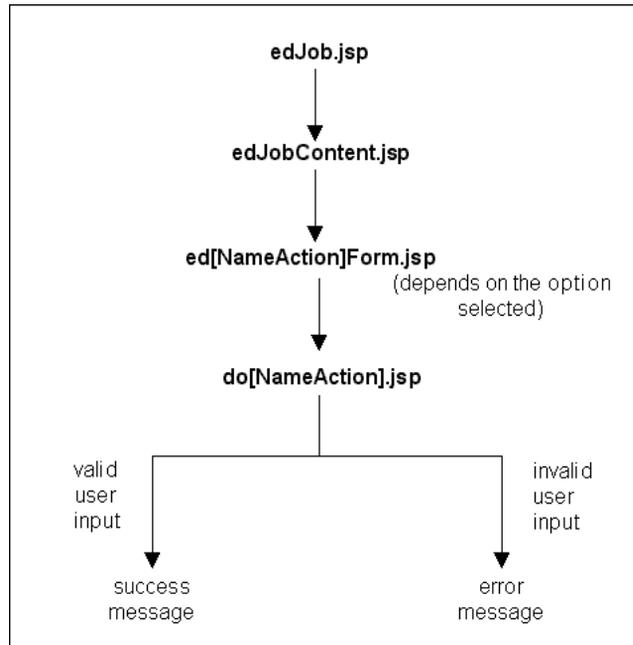
Action	Description	Action JSP File	Do Action JSP File
Edit folder content	Show, hide, or delete folders and services in a folder. The delete option takes precedence over the show or hide option.	edFolderListForm.jsp	doFolderContent.jsp

Configuring Jobs

Users can edit their jobs by clicking the Edit link corresponding to that job. The **edJob.jsp** frameset file displays the Edit User Jobs dialog. The dialog contains two frames:

- The top frame (**edJobMenu.jsp**) displays the options available to the user.
- The bottom frame (**insJob.jsp**) displays the instructions for all the menu options.

Depending on the option selected by the user, the input form in the bottom frame is displayed by the **edNameActionForm.jsp** file. When the user clicks the Finish button on the input form, the flow moves to a **doNameAction.jsp** file. The **doNameAction.jsp** file posts and processes the input values, and, depending on the validity of the input values, displays a success or an error message. The following displays the flow of control when a user edits a job.



The following lists the actions a user can perform in the Edit User Jobs dialog and the corresponding JSP files.

Action	Description	Action JSP File	Do Action JSP File
Rename job	Rename a job.	edServiceRenameForm.jsp	doServiceRename.jsp
Move job	Move a job to a new location. Users can choose either the up or down radio button to move the job up or down one level. Then, users can choose a location where they want to place the job.	edServiceMoveform.jsp	doServiceMove.jsp

Action	Description	Action JSP File	Do Action JSP File
Job content	Change the start date, start time, frequency, and agent for a job.	edJobForm.jsp	doJobCreate.jsp
Filter jobs	Specify a filtering condition for the job. Users select an output parameter from a drop-down list and add a filtering condition for that parameter.	edJobChangeForm.jsp	doJobChange.jsp

Configuring the User Profile

The **edUserPref.jsp** file is the frameset file that displays the Edit User Preferences dialog. This dialog consists of two frames:

- The top frame (**edUserPrefMenu.jsp**) displays the menu options available to the user.
- The bottom frame (**insUserPref.jsp**) displays the instructions for all the menu options.

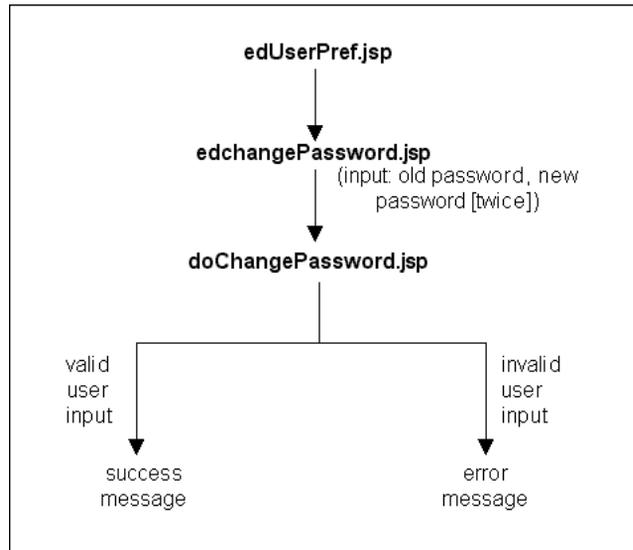
The Edit User Preferences dialog allows users to perform two types of actions:

- [Changing Password](#)
- [Configuring Agents](#)

Changing Password

The **edUserPref.jsp** file displays the Edit User Preferences dialog. Once the user selects the Change Password option, the flow moves to the **edPasswordChangeForm.jsp** file. This file displays the input form for retrieving the old and new password from the user. Once the user clicks the Finish button on the input form, the flow moves to the **doPasswordChange.jsp** file. Depending on the validity of user input (old and new passwords), a success or error message is

displayed. The following displays the flow of control when a user changes passwords.



Configuring Agents

Agents enable Portal-to-Go to transmit notifications to users. Notifications are alerts that are sent to a user’s device to notify the user of a specific event, such as a stock reaching a target price, in the case of a stock service, or a flight delay, in the case of a travel service.

The **edUserPref.jsp** file displays the Edit User Preferences dialog, which allows users to perform the following actions. These actions are displayed in the top frame:

- Create an agent
- Specify a default agent
- Edit an agent
- Delete an agent

Depending on the action selected by the user, an input form is displayed in the bottom frame by an **edNameActionForm.jsp** file. When the user clicks the Finish button on the input form, the flow moves to a **doNameAction.jsp** file. The **doNameAction.jsp** file posts and processes the input values, and, depending on the validity of the input values, displays a success or an error message.

The following displays the actions a user can perform to manipulate agent information and the corresponding JSP files.

Action	Description	Action JSP File	Do Action JSP File
Create agent	Create an agent. Users need to specify a unique name for the agent, its address, and the device type.	edAgentCreateForm.jsp	doAgentCreate.jsp
Default agent	Select an agent as the default agent.	edAgentSelectForm.jsp	doAgentChange.jsp
Edit agent	Edit an agent's properties.	edAgentSelectFom.jsp, edAgent.jsp	doAgentChange.jsp, doAgentEdit.jsp
Delete agent	Delete an agent.	edAgentSelectForm.jsp	doAgentChange.jsp

Debug Messages

A debug feature in the Personalization Portal puts detailed information in the **sys_panama.log** file. The .jsp file **initPRequest.jsp** initializes the **papzRequest** bean. The last argument of the initialize method call is true or false, which indicates whether you want the debug messages to appear in the log file or not.

The default value is set to false. To get detailed debug messages, set the value to true. Because the **initPRequest.jsp** file is a component file that is included in other container files, you should change the timestamp of the container file (for example, **PapzMain.jsp**). After you make these two changes, detailed debug messages, with the correct timestamp, will be printed in the log file.

National Language Support

This release of Portal-to-Go supports single-byte, multi-byte, and fixed-width encoding schemes which are based on national, international, and vendor-specific standards.

Overview

If the character set is single byte, and that character set includes only composite characters, the number of characters and the number of bytes are the same. If the character set is multi-byte, there is generally no such correspondence between the

number of characters and the number of bytes. A character can consist of one or more bytes, depending on the specific multi-byte encoding scheme.

A typical situation is when character elements are combined to form a single character. For example, in the Thai language, up to three separate character elements can be combined to form one character, and one Thai character would require up to 3 bytes when TH8TISASCII or another single-byte Thai character set is used. One Thai character would require up to 9 bytes when the UTF8 character set is used.

Multibyte Encoding Schemes

Multibyte encoding schemes are needed to support ideographic scripts used in Asian languages like Chinese or Japanese since these languages use thousands of characters. These schemes use either a fixed number of bytes to represent a character or a variable number of bytes per character.

Fixed-width Encoding Schemes

In a fixed-width multibyte encoding scheme, each character is represented by a fixed number of n bytes, where n is greater than or equal to two.

Variable-width Encoding Schemes

A variable-width encoding scheme uses one or more bytes to represent a single character. Some multibyte encoding schemes use certain bits to indicate the number of bytes that represent a character. For example, if two bytes is the maximum number of bytes used to represent a character, the most significant bit can be toggled to indicate whether that byte is part of a single-byte character or the first byte of a double-byte character. In other schemes, control codes differentiate single-byte from double-byte characters. Another possibility is that a shift-out code is used to indicate that the subsequent bytes are double-byte characters until a shift-in code is encountered.

Setting the Multi-Byte Encoding for the Personalization Portal

The portal gets the encoding for the text of the site from the setting in the PAPZ logical device, which is in the repository. The default encoding is ISO-8559-1, which is the standard for the Western European languages. The portal sets the content for each page with the encoding specified by the logical device. To change the default encoding to multi-byte encoding, which enables the display of various Asian languages, click PAPZ under Logical Devices in the Service Designer and change the encoding according to the IANA standards for your particular language.

Localization

To localize the display of information, you can modify the static text for the JSP pages in the Service Designer and the Personalization Portal.

Service Designer

To localize the Service Designer, you can modify the following local strings resource files:

- **PASMEException.properties** (error messages)
- **PASMSuggestion.properties** (suggestions for some of the error messages)
- **Resources.properties** (labels for the Service Designer GUI)

These files are located in the directory:

...\Service Designer\classes\oracle\panama\pasm

Personalization Portal

To localize the Personalization Portal, you can modify the following local strings resource files:

- oracle\panama\personalization\jsp\resources\LocalStrings.properties
- oracle\panama\personalization\resources\LocalStrings.properties

These directories are extracted when you unzip **panama_papz.zip**.

Working with Portal-to-Go XML

This document describes the XML formats Portal-to-Go uses to represent service content and its internal objects. Topics include:

- [Why XML?](#)
- [Oracle XML Parser](#)
- [Portal-to-Go XML Formats](#)
- [Adapter Result Format](#)
- [Simple Result Format](#)
- [The Portal-to-Go Repository](#)
- [XML Tools](#)

Why XML?

Extensible Markup Language (XML) is a standards-based language for creating structured documents. In an XML document, markup tags describe the content of the document, rather than its visual presentation:

```
<account_dept>
  <employee>
    <emp_name>Scott</emp_name>
    <emp_ID>20</emp_ID>
  </employee>
  <employee>
    <emp_name>Laura</emp_name>
    <emp_ID>30</emp_ID>
  </employee>
</account_dept>
```

Because it isolates content from presentation, XML is an ideal language for exchanging data between diverse platforms. Transformers, in the form of XSLT stylesheets or Java classes, render the data in the format best suited to a particular platform.

Oracle XML Parser

Portal-to-Go uses the Oracle XML Parser for Java, version 2.0.2.8. The parser supports the W3C XML 1.0 Standard Recommendation. While the parser supports various character encoding standards, Portal-to-Go uses UTF-8 character encoding.

Version 2.0.2.8 of the parser requires a version attribute in the XSL stylesheet, while prior versions do not. If you have created your own result and/or device transformers, you have to include the XML version number in the XSL stylesheet.

For example:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Without the version attribute, you will receive a runtime error in the Portal-to-Go log file in the form of "missing version attribute from the stylesheet."

The Oracle XML Parser exposes two interfaces:

- The Document Object Model (DOM) Level 1 API, as specified in the W3C Recommendation
- The Simple API for XML 1.0 (SAX) interface

To improve performance and usability, the APIs reside in a single archive. The Oracle XML Parser is provided as a **.jar** file comprising the following four packages:

- `oracle.xml.parser`
- `org.w3c.dom`
- `org.xml.sax`
- `org.xml.sax.helpers`

In addition to the standard interfaces, the parser provides numerous extensions. For more information on the extension, see the Portal-to-Go API specifications.

Portal-to-Go XML Formats

In Portal-to-Go, XML serves two primary functions: to define the format of service content and to define repository objects. The content formats include the following:

- Simple Result
- Adapter Result

The Simple Result DTD defines the format of deliverable service content. Adapter Result is an intermediary format for service content.

The object formats include:

- Provisioning DTD
- Repository XML

The Provisioning DTD defines the structure of user objects. You can use the Provisioning DTD to transfer user information between Portal-to-Go and an external provisioning system.

Repository XML is not defined by a DTD, since repository objects can have extended attributes which cannot be validated. An extended attribute can be an XML document, such as XSLT, or a Java program. Transformer and adapter objects have extended attributes.

Portal-to-Go XML is intended to be extensible. You can modify the DTDs that Portal-to-Go provides, or create your own. This enables you to implement specialized functionality or to customize Portal-to-Go object characteristics.

Symbols

This document contains references for the Portal-to-Go XML formats. The references list the attributes and subelements for every element in the Simple Result and Provisioning DTDs. It uses the following DTD symbols to indicate the usage of subelements.

Symbol	Meaning
+	Indicates that a subelement must appear at least once within the parent element. If the plus sign appears after a parenthetical grouping of subelements, then at least one of the listed subelements must appear within the parent element.
?	Indicates that a subelement may appear only once or not at all.

Adapter Result Format

The Adapter Result (also called raw result) format is a simplified, user interface-independent content format. Adapter Result documents must be converted to Simple Result format prior to conversion to their final target format. This intermediary step, called preprocessing, enables you to apply a service-specific transformation to the content returned by an adapter. Result transformers convert Adapter Result documents. Adapter Result format provides an efficient means of exchanging data between chained services.

Note: Currently, only services based on the Web Integration adapter can use the Adapter Result format.

Portal-to-Go does not specify an Adapter Result DTD. You can define the content any way you like. While the content is not validated by Portal-to-Go, it must be well-formed. The only element Portal-to-Go requires in Adapter Result content is the `<AdapterResult>` tag. The remaining structure of the content is important only to the transformer that processes it. For example:

```
<AdapterResult>
  <Value name="target">
    /parm/epayment/amex
  </Value>
  <Value name="creditcard">
    Please enter your credit card no:
  </Value>
  <Value name="expdate">
    Please enter expiration date:
  </Value>
</AdapterResult>
```

The sample uses a single element type, the `Value` tag. The first `Value` element contains a link to another service in the Portal-to-Go repository tree (using JNDI-naming format).

A result transformer converts the Adapter Result content as follows:

```
<SimpleResult>
  <SimpleForm link="/parm/epayment/amex">
    <SimpleFormItem name="creditcard" description="Please enter
      your credit card no:"/>
    <SimpleFormItem name="expdate" description="Please enter
```

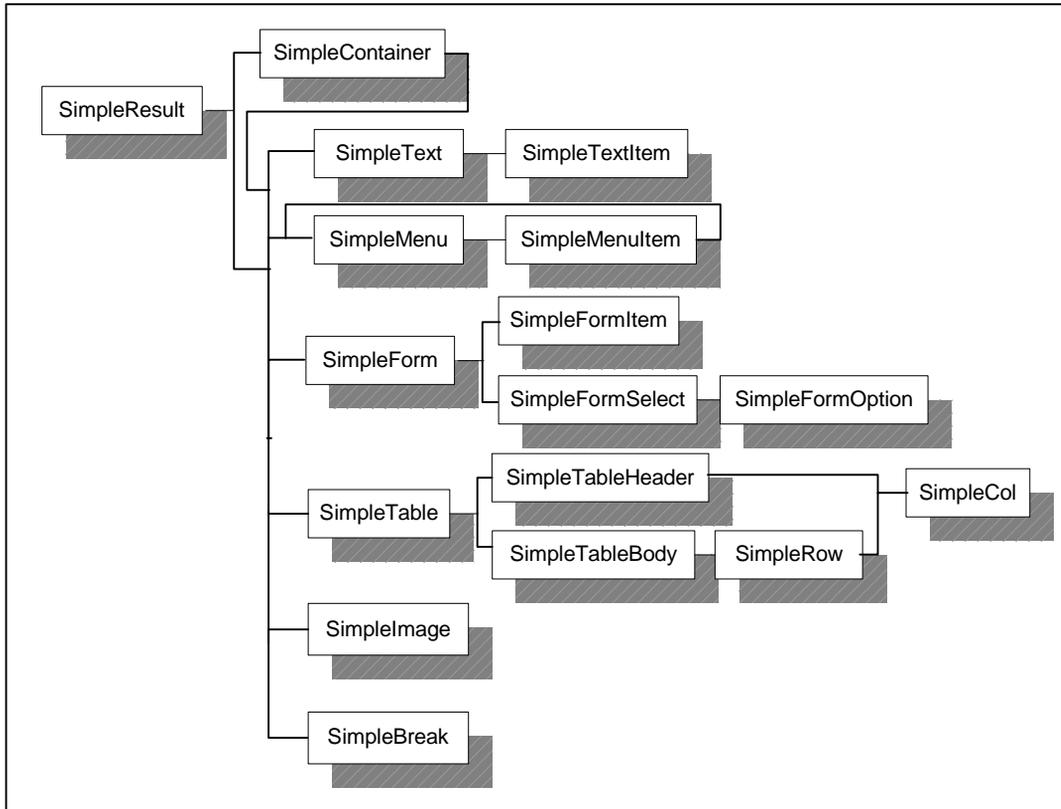
```
expiration date:"/>  
</SimpleForm>  
</SimpleResult>
```

Simple Result Format

The elements in the Simple Result DTD represent the elements of an abstract user interface. These include text items, menus, forms, and tables. When converting source content to Simple Result format, adapters map the source content to the appropriate Simple Result element. Likewise, when converting the content from Simple Result format to the target format, transformers map the Simple Result elements to the appropriate elements in the target format.

Content Model

The Simple Result DTD defines the following content model.



[Appendix A, "Simple Result DTD Notes"](#), describes the content of the Simple Result DTD in detail and in a clear format.

Element Reference

This section describes the components of the Simple Result DTD.

SimpleResult Element

The `SimpleResult` element is the root element of a Portal-to-Go service request. This element contains the actual content delivered to the end user.

Attributes

name

The name of the element. This is an optional attribute.

title

The title of the element. This is an optional attribute.

link

A link for this element. This is an implied link and should not be used for mandatory links in an element. It should be used for "see also" type links only.

Subelements

(SimpleContainer

SimpleText

SimpleMenu

SimpleForm

SimpleTable

SimpleImage

SimpleBreak

SimplePhone

SimpleEmail

SimpleHref

SimpleHelp)+

SimpleContainer Element

The `SimpleContainer` element is used as a logical container for one or more elements.

Attributes

name

The name of the element. This is an optional attribute.

title

The title of the element. This is an optional attribute.

link

A link for this element. This is an implied link and should not be used for mandatory links in an element. It should be used for "see also" type links only.

Subelements

(SimpleText
SimpleMenu
SimpleForm
SimpleTable
SimpleImage
SimpleBreak)+

SimpleText Element

The `SimpleText` element contains one or more blocks of plain text.

Attributes

name

The name of the element. This is an optional attribute.

title

The title of the element. This is an optional attribute.

link

A link for this element. This is an implied link and should not be used for mandatory links in an element. It should be used for "see also" type links only.

Subelements

SimpleTextItem+

SimpleTextItem Element

The `SimpleTextItem` element represents a block of plain text, typically a single paragraph.

Attributes

name

The name of the element. This is an optional attribute.

title

The title of the element. This is an optional attribute.

link

A link for this element. This is an implied link and should not be used for mandatory links in an element. It should be used for "see also" type links only.

Contents

#PCDATA

SimpleMenu Element

The `SimpleMenu` element represents a simple menu with selectable menu items.

Attributes

name

The name of the element. This is an optional attribute.

title

The title of the element. This is an optional attribute.

link

A link for this element. This is an implied link and should not be used for mandatory links in an element. It should be used for "see also" type links only.

Subelements

`SimpleMenuItem+`

SimpleMenuItem Element

The `SimpleMenuItem` element represents a single, selectable option in a menu. The option may be another `SimpleMenu` element.

Attributes

name

The name of the element. This is an optional attribute.

title

The title of the element. This is an optional attribute.

link

A link for this element. This is an implied link and should not be used for mandatory links in an element. It should be used for "see also" type links only.

target

The link target for this item.

section

Used by the WIDL adapter. This is the section identifier within the WIDL file.

separator

A separator to precede or follow a menu item. Possible values:

- before
- after
- none

The default is none.

SimpleForm Element

The `SimpleForm` element displays one or more input fields.

Attributes

name

The name of the element. This is an optional attribute.

title

The title of the element. This is an optional attribute.

link

A link for this element. This is an implied link and should not be used for mandatory links in an element. It should be used for "see also" type links only.

target

The link target for this item.

section

Used by the WIDL adapter. This is the section identifier within the WIDL file.

Subelements

```
(SimpleFormItem  
 SimpleFormSelect)+
```

SimpleFormItem Element

The `SimpleFormItem` element is a single input item in a form. The content of this element, which is in parsable character format, specifies default values for the form item.

Attributes**name**

The name of the element. This is an optional attribute.

title

The title of the element. This is an optional attribute.

link

A link for this element. This is an implied link and should not be used for mandatory links in an element. It should be used for "see also" type links only.

default

Provides a default value for optional fields. The default value is used only if the field is empty.

mandatory

Indicates whether the form item is mandatory. Possible values are:

- `yes`
- `no`

The default is `no`.

maxLength

Specifies a maximum input length.

Contents

#PCDATA

SimpleFormSelect Element

The `SimpleFormSelect` element is a selectable option menu in a form. The content of this element, which is in parsable character format, specifies default values for the form item.

Attributes

name

The name of the element. This is an optional attribute.

title

The title of the element. This is an optional attribute.

link

A link for this element. This is an implied link and should not be used for mandatory links in an element. It should be used for "see also" type links only.

Subelements

`SimpleFormOption+`

SimpleFormOption Element

The `SimpleFormOption` element is an item in a selectable option menu. The content of this element, which is in parsable character format, specifies default values for the form item.

Attributes

name

The name of the element. This is an optional attribute.

title

The title of the element. This is an optional attribute.

link

A link for this element. This is an implied link and should not be used for mandatory links in an element. It should be used for "see also" type links only.

value

The actual value, as character data, assigned when the user selects this option.

Contents

#PCDATA

The contents of this element comprise the text of the option.

SimpleTable Element

The `SimpleTable` element represents simple tabular data.

Attributes**name**

The name of the element. This is an optional attribute.

title

The title of the element. This is an optional attribute.

link

A link for this element. This is an implied link and should not be used for mandatory links in an element. It should be used for "see also" type links only.

Subelements

`SimpleTableHeader?`

`SimpleTableBody`

SimpleTableHeader Element

The `SimpleTableHeader` element represents column headings.

Attributes**name**

The name of the element. This is an optional attribute.

title

The title of the element. This is an optional attribute.

link

A link for this element. This is an implied link and should not be used for mandatory links in an element. It should be used for "see also" type links only.

Subelements

SimpleCol+

SimpleTableBody Element

The SimpleTableBody element contains the actual tabular data.

Attributes

name

The name of the element. This is an optional attribute.

title

The title of the element. This is an optional attribute.

link

A link for this element. This is an implied link and should not be used for mandatory links in an element. It should be used for "see also" type links only.

Subelements

SimpleRow+

SimpleRow Element

The SimpleRow element contains a single row of data.

Attributes

This element includes the general attributes.

Subelements

SimpleCol+

SimpleCol Element

The SimpleCol element represents a single table cell. It stores a single value for a table cell.

Attributes

name

The name of the element. This is an optional attribute.

title

The title of the element. This is an optional attribute.

link

A link for this element. This is an implied link and should not be used for mandatory links in an element. It should be used for "see also" type links only.

Contents

#PCDATA

SimpleImage Element

The `SimpleImage` element references an external image.

Attributes

target

The link target for the image.

SimpleBreak Element

The `SimpleBreak` element provides a logical break in the layout. This is an empty element.

SimplePhone Element

The `SimplePhone` element references a phone number.

Attributes

target

The phone number

SimpleEmail Element

The `SimpleEmail` element references an email address.

Attributes

target

The email address

SimpleHref Element

The `SimpleHref` element provides an anchor to another resource.

Attributes

target

The link target for the reference.

SimpleHelp

The `SimpleHelp` element indicates a help text.

Note: For more information, see [Appendix A, "Simple Result DTD Notes,"](#) .

The Portal-to-Go Repository

The Portal-to-Go repository holds the persistent Portal-to-Go objects, such as end users, user groups, services, adapters, and transformers. Internally, Portal-to-Go stores these objects as database objects in an Oracle8i database. In most cases, you create or modify repository objects using the Service Designer. However, you can also export, modify, and import the repository as an XML file, using utilities provided by Portal-to-Go. To use these utilities, you must understand the XML structure of the repository, as described in this section.

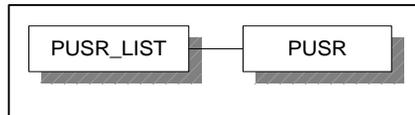
Portal-to-Go provides an XML editor that you can use to modify the XML definitions of objects directly in the repository. You can also modify objects in an XML file. To do so, you export the repository using `LoadXml` or the `download` utility, and edit the resulting XML file. When finished, you re-import the file into the repository, committing your changes. See "[XML Tools](#)" for more information on using the XML Editor or the `LoadXML` utility.

Repository XML enables you to distribute populated repositories easily. It also enables you to access the repository programmatically, and to integrate Portal-to-Go with other systems.

Important: Changes made directly to the repository XML are not validated by the Service Designer. Use caution to ensure that the changes you make do not cause conflicts or errors.

Provisioning DTD

The Provisioning DTD defines the XML format for Portal-to-Go users. You can use it to import users into the repository programmatically, or to import a large number of users at once. As illustrated in the following content model, the Provisioning DTD declares only two elements:



PanamaObjects Element

The `PanamaObjects` element is the root provisioning document element.

Subelements

`PUSR_LIST`

PUSR_LIST Element

The Portal-to-Go user list.

Subelements

`PUSR+`

PUSR Element

The `PUSR` element represents a Portal-to-Go user.

Attributes

externalId

A user's unique identifier in an external provisioning system. For example, this may be a telephone number or an account number. You can use this attribute to tie a Portal-to-Go user to another account. This is an optional attribute.

display name

User's display name. This is an optional attribute.

enabled

Indicates whether the user is enabled. Possible values are:

- True
- False

password

The user's password. In exported repositories, this appears in encrypted form. For uploading, this can be either an encrypted or plain text password. If the string is greater in length than 32 characters, and all uppercase, Portal-to-Go assumes that it is an encrypted password.

administrator

Indicates whether the user has administrator privileges. Possible values are:

- True
- False

designer

Indicates whether the user has designer privileges. Possible values are:

- True
- False

name

The Portal-to-Go user identifier.

anonymous

Indicates whether the user is anonymous. Possible values are:

- True
- False

userRoot

The users home folder.

group

The name of the group to which the user belongs.

Repository XML Reference

The XML file you import into the repository must contain appropriately formatted repository objects. Since objects in the repository, such as transformers, can have extended attributes, Portal-to-Go does not validate repository XML documents. Therefore, Portal-to-Go does not define a Repository DTD.

Your XML definition should use the following elements for repository objects.

PanamaObjects Element

The `PanamaObjects` element is the root element.

Subelements

`PGRP_LIST`

`PUSR_LIST`

`TRAN_LIST`

PGRP_LIST Element

The Portal-to-Go group list.

Attributes

name

The name of the element. This is an optional attribute.

title

The title of the element. This is an optional attribute.

link

A link for this element. This is an implied link and should not be used for mandatory links in an element. It should be used for "see also" type links only.

Subelements

PGRP

PGRP Element

The PGRP element represents a Portal-to-Go user group.

Attributes

name

The name of the element. This is an optional attribute.

PUSR_LIST Element

The Portal-to-Go user list.

Subelements

PUSR

PUSR Element

The PUSR element represents a Portal-to-Go user.

Attributes

externalId

External identifier. This is an optional attribute.

display name

User's display name. This is an optional attribute.

enabled

Indicates whether the user is enabled. Possible values are:

- True
- False

password

The user's password. In exported repositories, this appears in encrypted form. For uploading, this can be either an encrypted or plain text password. If the string is greater in length than 32 characters, and all uppercase, Portal-to-Go assumes that it is an encrypted password.

administrator

Indicates whether the user has administrator privileges. Possible values are:

- True
- False

designer

Indicates whether the user has designer privileges. Possible values are:

- True
- False

name

The user name.

anonymous

Indicates whether the user is anonymous. Possible values are:

- True
- False

userRoot

The user's home folder.

group

The name of the group to which the user belongs.

TRAN_LIST Element

The Portal-to-Go transformer list.

Subelements

XTRA

JTRA

XTRA Element

The XTRA element represents an XSLT Portal-to-Go transformer.

Attributes

name

The name of the element.

Subelements

EXT_ATTR

EXT_ATTR Element

Extended attribute. The actual XML element that comprises the XSLT implementation.

JTRA Element

The JTRA element represents a Portal-to-Go Java transformer.

name

The name of the element.

className

The complete class name including attributes.

LDEV_LIST Element

The Portal-to-Go logical device list.

Subelements

LDEV

LDEV Element

The LDEV element represents a Portal-to-Go logical device.

Attributes

needsURLCaching

Indicates whether the logical device needs URL caching. If true, URL-encoding request parameters are cached on the server and only small pointers are encoded within the URL that is sent to the logical device. Use this option for devices with limited memory. Possible values are:

- True

- False

name

The name of the logical device.

mimeType

MIME type.

encoding

Character encoding format, such as UTF-8 or ASCII.

defaultTransformer

The default transformer used by the logical device.

prolog

A string inserted in front of the output from the transformer, such as an XML document type.

ADAP_LIST Element

The Portal-to-Go adapter list.

Subelements

ADAP

ADAP Element

The ADAP element represents a Portal-to-Go adapter.

valid

Indicates whether the adapter is valid (accessible). Possible values are:

- True
- False

name

The adapter name.

className

The Java class name.

PSRV_LIST Element

The Portal-to-Go service object list.

Subelements

FOLD

LINK

MAST

BOMA

FOLD Element

The FOLD element represents a Portal-to-Go service folder.

Attributes

url

The location of the service folder, relative to the root URL of the service tree. This attribute is set by the Portal-to-Go system. It cannot be edited.

name

The service folder name.

visible

Indicates whether the folder is visible. Possible values are:

- True
- False

valid

Indicates whether the folder is valid (available). Possible values are:

- True
- False

folder

The master folder where the service folder is located.

owner

The user name of the folder owner.

areald

A value which determines whether a folder is visible, depending on a user's physical location.

group

The user group that is allowed to access the folder. This is an optional attribute.

LINK Element

The `LINK` element represents a Portal-to-Go service link.

Attributes**url**

The location of the service link, relative to the root URL of the service tree. This attribute is set by the Portal-to-Go system. It can not be edited.

name

The service link name.

visible

Indicates whether the link is visible. Possible values are:

- True
- False

valid

Indicates whether the link is valid (accessible). Possible values are:

- True
- False

folder

The folder where the link is located.

owner

The user name of the link owner.

service

Name of the linked service.

MAST Element

The **MAST** element represents a Portal-to-Go master service.

Attributes

url

The location of the master service, relative to the root URL of the service tree. This attribute is set by the Portal-to-Go system. It can not be edited.

name

The master service name.

visible

Indicates whether the master service is visible. Possible values are:

- True
- False

valid

Indicates whether the master service is valid (accessible). Possible values are:

- True
- False

folder

The folder where the master service is located.

owner

The user name of the master service owner.

areaID

A value identifying the geographic location of the user.

cost

The amount charged for using the service.

adapter

The adapter used by the master service.

BOMA Element

The **BOMA** element represents a bookmark.

Attributes

url

The location of the bookmark, relative to the root URL of the service tree. This attribute is set by the Portal-to-Go system. It can not be edited.

name

The bookmark name.

visible

Indicates whether the bookmark is visible. Possible values are:

- True
- False

valid

Indicates whether the bookmark is valid (accessible). Possible values are:

- True
- False

folder

The folder where the bookmark is located.

href

The URL of the external datasource.

AGEN_LIST Element

The Portal-to-Go user agent list.

Subelements

AGEN

AGEN Element

The AGEN element represents a Portal-to-Go user agent.

address

The address of the user agent.

name

The name of the user agent.

user

The Portal-to-Go user associated with the user agent.

logicalDevice

The logical device.

uniqueName

An identifying attribute set by Portal-to-Go.

XML Tools

Portal-to-Go includes the following tools for working directly with the XML interface:

- [XML Editor](#)
- [LoadXml](#)
- [Upload and Download Utilities](#)

XML Editor

The Portal-to-Go XML Editor is a text editor that allows you to create, modify, and delete repository objects using the XML object interface. The XML Editor provides a graphical navigation tree for accessing objects in the repository.

Important: The XML Editor does not validate the changes you make to repository objects. Use caution to ensure that the changes you make do not create conflicts or errors in the repository. You should back up the repository before modifying it with the XML Editor.

Requirements

To use the XML Editor, you must have an RMI server running on the Portal-to-Go server.

Synopsis

```
oracle.panama.util.client.xmleditor.XmlEditor
```

Using the XML Editor

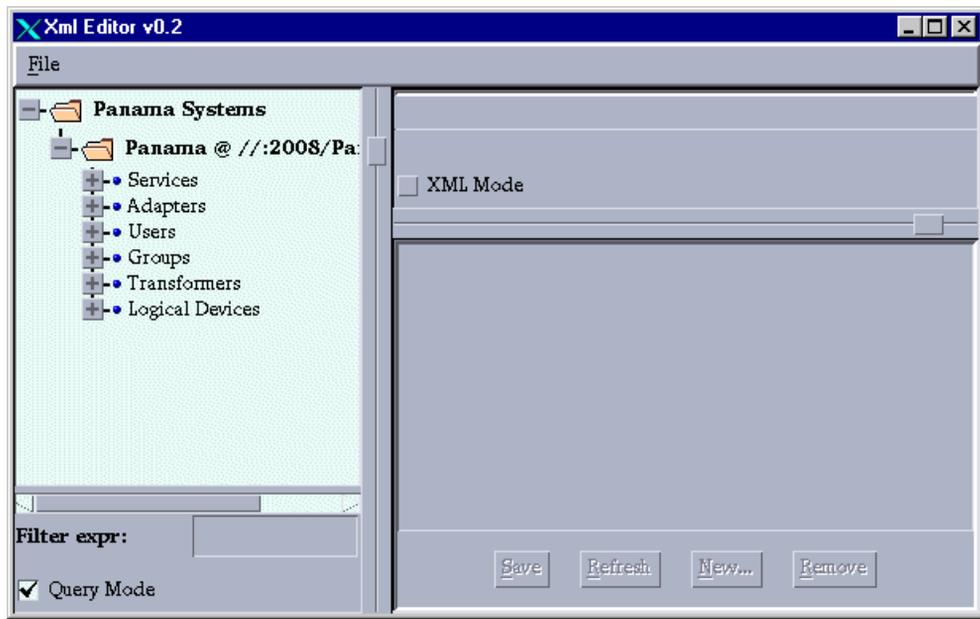
To start the XML Editor:

1. Log on to your Portal-to-Go host.
2. From the Unix command line enter the following:

```
Java oracle.panama.util.client.xmleditor.XmlEditor
```
3. The Logon dialog appears. Enter the following information and click OK.

Field	Description
User Name	The user name of the Portal-to-Go host application server.
Password	The password of the Portal-to-Go host application server.
URL	The URL of the Portal-to-Go RMI server. The format is: <i>//host:port/servername</i> The host is the name of the machine that hosts the Portal-to-Go server. The port and server name identify the Portal-to-Go RMI server. The default values are: <ul style="list-style-type: none">■ Port: 2008■ Server name: PanamaServer You can verify and configure these settings in the properties file Rmi.properties .

The XML Editor Screen appears.



To edit a repository object:

1. Select the object from the object tree in the left panel of the screen. The object's XML definition appears in the text box to the right.
2. Make the desired changes to the object definition.
3. Click the Save button to save the changes to the object.

To remove an object from the repository:

1. Select the object from the object tree. The object's XML definition appears in the text box.
2. Click the remove button. The object is removed from the repository.

Query Mode

When the query mode check box is selected, the XML Editor refreshes object definitions from the repository database each time an object is selected. This is recommended, especially when more than one user may be editing repository objects. You can also use the Refresh button to refresh an object definition from the repository database.

LoadXml

The `LoadXML` utility allows you to download and upload Portal-to-Go repository objects as XML files.

`LoadXml` reads from `stdin` and writes to `stdout`. All logging and error messages are written to `stderr`. The XML in the file you import with `LoadXml` must conform to the Repository XML.

The upload function performs the following:

1. Checks for the objects in the repository by logical unique name.
2. Loads all dependencies.
3. If the objects exist in the repository, `LoadXml` updates the objects.
4. If the objects do not exist, `LoadXml` creates new objects.
5. After each object type is successfully loaded, `LoadXml` performs a commit. The commit includes all referenced objects (dependencies).

In the unload XML result all objects have an attribute called `_objectId`, this is the system unique object key. You must look up objects by unique name attribute and not the object key. If you start the program without giving an option, all options are listed.

`LoadXML` imports and exports the repository identified by the database connect string in the **System.properties** file. In the development environment, this file is located in the `classes/oracle/panama/core/admin` directory.

Portal-to-Go does not validate the XML file you import into the repository with `LoadXml`. To avoid errors, work in an XML file that you have exported from the repository. This gives you a “known good” Repository XML framework for adding, removing, and modifying individual elements.

Synopsis

```
oracle.panama.core.util.LoadXml [-l username/password] [-x[adgnstu] [expr]] [-c# [-p]]
```

Options

The `LoadXml` utility takes the following options:

Option	Description
<code>-l usr/pwd</code>	Log on to Portal-to-Go using a user name and password. If no administrator is defined in the system, the program allows any user to log on. Otherwise, the user must be an administrator to log on.
<code>-x</code>	Unload all repository data to <code>stdout</code> . Can be filtered by adding more options.
<code>-a</code>	Adapter filter. Used with the <code>-x</code> option.
<code>-d</code>	Logical device filter. Used with the <code>-x</code> option.
<code>-g</code>	Group filter. Used with the <code>-x</code> option.
<code>-n</code>	Agent filter. Used with the <code>-x</code> option.
<code>-s</code>	Service filter. Used with the <code>-x</code> option.
<code>-t</code>	Transformer filter. Used with the <code>-x</code> option.
<code>-u</code>	User filter. Used with the <code>-x</code> option.
<code>expr</code>	Name expression filter. This option can include wildcards, such as: <code>[*%?_]</code> . Used with the <code>-x</code> option.
<code>-c#</code>	Upload repository data read from <code>stdin</code> . The argument <code>#</code> is a number that, if set, causes a commit after the specified number of objects are uploaded. An argument of <code>0</code> causes a commit after a complete load of the XML data.
<code>-p</code>	This option activates provisioning of user data when uploading. This mode is only used with the <code>-c</code> option. The provisioning upload handles the provisioning DTD, and makes it possible to create, enable, disable, and remove users in the repository. This mode always creates new users. If the <code>createUserRoot</code> attribute is set to <code>YES</code> , <code>LoadXml</code> creates a root folder for each user. The provisioning upload uses streaming to load users, and therefore does not resolve dependencies.

Unload Example

In this example, XML data is written to standard output.

```
java oracle.panama.core.util.LoadXml -l adm/adm -x > outputfile.xml
```

Upload Example

In this example, XML data is read from standard input. It must contain all referenced objects.

```
prompt$ java oracle.panama.core.util.LoadXml -l adm/adm -c0 < inputfile.xml
```

Upload and Download Utilities

You can use the `upload` and `download` utilities to import and export the Portal-to-Go repository as an XML file. These utilities invoke `LoadXml`. They are located in the `ServiceDesigner/sample` directory of your development environment. You invoke `upload` from a command prompt as follows:

```
upload.bat repository.xml
```

This loads the contents of the file **repository.xml** into the repository. It accesses the repository specified by the connect string in the client-side **System.properties** file.

To download a repository:

```
download.bat repository.xml
```

This places the contents of the repository in a file named **repository.xml**.

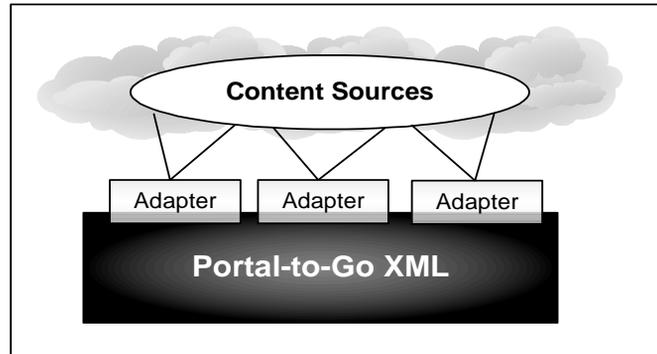
This document describes how to create and manage Java classes and stylesheets that implement Portal-to-Go adapters. Topics include:

- [What Is a Portal-to-Go Adapter?](#)
- [Creating Adapters](#)
- [Importing an Adapter into the Repository](#)
- [Extending the Stripper Adapter](#)

What Is a Portal-to-Go Adapter?

A Portal-to-Go adapter is a Java application that retrieves data from an external source and renders it in Portal-to-Go XML. When invoked by a master service, an adapter returns an XML document that contains the service content.

In the Portal-to-Go architecture, adapters constitute the interface between the Portal-to-Go server and the content source.



Portal-to-Go adapters primarily return content. But they can also perform other functions. For example, the Provisioning adapter, which is included in the Portal-to-Go initial repository, manages Portal-to-Go users. You can use it to create, remove, and modify users.

Using Portal-to-Go Adapters

To use a Portal-to-Go adapter, you create a master service that invokes the adapter. Master services encapsulate the primary Portal-to-Go functionality; they make information and applications available to the end users.

Portal-to-Go provides adapters that retrieve Web and database content. [Chapter 5, "Portal-to-Go Services"](#) describes the adapters provided by Portal-to-Go, and how to create services that use them.

Implementing Portal-to-Go Adapters

You create your own Java adapters for Portal-to-Go by implementing the `Adapter` interface. The adapter you create can be simple or very complex. It can be a self-contained application or, as in the case of the Web Integration adapter, it can invoke external applications. How the adapter retrieves information is transparent to Portal-to-Go. It must simply expose the entry points and return the value types that Portal-to-Go expects.

Content Formats

Adapters return content that conforms to a Portal-to-Go content DTD. In most cases, adapters return content that conforms to the Simple Result DTD. The Simple

Result DTD contains elements that represent the components of an abstract UI, such as tables, forms, and menus.

Adapters can also return content in Adapter Result format. Adapter Result format is a intermediary, user interface-independent content format. You can use it to pass raw data between Portal-to-Go components, such as chained services. Since device transformers operate only on Simple Result format, a result transformer associated with a master service must convert an Adapter Result document to Simple Result format before the content can be delivered.

For more information on Portal-to-Go XML, see [Chapter 8, "Working with Portal-to-Go XML."](#)

Creating Adapters

Generally, an adapter performs the following functions:

1. Connects to a data source.
2. Retrieves content.
3. Converts the content to a Portal-to-Go XML format.

The adapter should know the content source format and how to map it to a Portal-to-Go results format.

Follow these rules to create an adapter:

1. Implement the `Adapter` interface.
2. Provide an empty, default constructor.
3. Make the class thread-safe.
4. Implement the following methods in the `Adapter` class:
 - `init()`
 - `invoke()`
 - `getInitDefinition()`
 - `getAdapterDefinition()`

The master service calls the `init()` function once, the first time the adapter is invoked. The `init()` function should perform initialization functions for the adapter. The master service calls the `invoke()` function every time the adapter is invoked. The `getInitDefinition()` and `getAdapterDefinition()` methods enable Portal-to-Go to create an adapter definition for an adapter.

Portal-to-Go calls the methods in the following order:

1. Default constructor
2. `getInitDefinition()`
3. `init()`
4. `getAdapterDefinition()`
5. `invoke()`

The `invoke()` method is then called again every time the adapter is invoked.

Adapter Definition

An adapter definition defines the parameters used by an adapter. To create an adapter definition, you instantiate the `AdapterDefinition` interface. Each adapter has two adapter definitions: one for the input parameters required to initialize the adapter, and one for all adapter parameters—initializing, runtime, and output.

To make its parameters available to master services, each adapter must implement the following methods.

Method	Description
<code>getInitDefinition()</code>	This method should return the initial <code>AdapterDefinition</code> object, which defines the parameters required to initialize the adapter.
<code>getAdapterDefinition()</code>	This method can only be called on an initialized adapter. It should return the complete <code>AdapterDefinition</code> object, which defines the initial, input, and output parameters.

Sample Adapter Class

The following sample Java class is a complete, but minimal, adapter implementation. It greets a user by name. It takes a single initialization parameter, the string used for the greeting, and a single input parameter, the name of the user.

```
// Copyright (c) 1999, 2000 by Oracle Corporation, all rights reserved.
//
// Title: HelloAdapter
// Usage: Demonstrate Panama Adapter with init/input parameters

import org.w3c.dom.Element;
```

```
import org.w3c.dom.Document;
import org.w3c.dom.Text;

import oracle.panama.Argument;
import oracle.panama.Arguments;
import oracle.panama.ArgumentType;
import oracle.panama.ServiceRequest;
import oracle.panama.adapter.Adapter;
import oracle.panama.adapter.AdapterDefinition;
import oracle.panama.adapter.AdapterException;
import oracle.panama.adapter.AdapterHelper;
import oracle.panama.core.util.XMLUtil;
import oracle.panama.core.xml.XML;

public class HelloAdapter implements Adapter {

    private boolean initialized = false;

    /** Greeting phrase, default is Hello */
    private String greeting = "Hello";

    /** Init parameter name for the greeting phrase */
    public static final String GREETING = "greeting";

    /** Input parameter name for the name to greet */
    public static final String NAME = "name";

    /** This is the initialization method, called once when the adapter
     *  is instantiated for the very first time. Whatever you do here
     *  should be synchronized to ensure that things are not initialized
     *  twice.
     *  @param args The parameters needed for initialization
     */
    public void init (Arguments args) throws AdapterException {
        synchronized (this) {
            initialized = true;
            greeting = args.getInputValue( GREETING );
        }
    }

    /**
     * This is the main request handler, invoked each time a client makes
     * a request. This is a simple example that is capable of returning
     * output according to Portal-to-Go simpleresult.dtd.
     * @param sr The ServiceRequest object
     */
}
```

```

* @return Element The XML element holding the result
* @exception Exception Not used here.
*/
public Element invoke (ServiceRequest sr) throws AdapterException {

    // This adapter returns this XML element:
    // <SimpleResult>
    //     <SimpleText title="Oracle Portal-to-Go Server Hello Sample">
    //         <SimpleTextItem name="message" title="Portal-to-Go says:">
    //             <greeting> <name>!
    //         </SimpleTextItem>
    //     </SimpleText>
    // </SimpleResult>

    // Create result element
    Document doc = sr.getOwnerDocument();
    Element result = XML.makeElement(doc, "SimpleResult");

    // Create SimpleText element
    Element st = XML.makeElement(result, "SimpleText");
    st.setAttribute ("title", "Oracle Portal-to-Go Server HelloAdapter
Sample");
    result.appendChild (st);

    // Create SimpleTextItem element
    Element sti = XML.makeElement(st, "SimpleTextItem");
    sti.setAttribute ("name", "message");
    sti.setAttribute ("title", "Portal-to-Go says:");
    st.appendChild (sti);

    // Create actual text
    String name = sr.getArguments().getInputValue( NAME );
    Text txt = XML.makeText(sti, greeting + " " + name + "!");
    sti.appendChild (txt);

    return result;
}

// The following method is required. If there are any initialization
// parameters for this adapter they would be set here.
private AdapterDefinition initDef = null;
public AdapterDefinition getInitDefinition() {
if (initDef == null) {
    synchronized (this) {
if (initDef == null) {

```

```

        initDef = AdapterHelper.createAdapterDefinition();
        // Any init parameters would be set here..
        initDef.createInit( ArgumentType.SINGLE_LINE, GREETING, "Greeting phrase",
null );
    }
}
return initDef;
}

    private AdapterDefinition adpDef = null;
    public AdapterDefinition getAdapterDefinition() throws AdapterException {
if (adpDef == null ) {
    synchronized (this) {
if (adpDef == null) {
    if (initDef == null)
throw new AdapterException ("Adapter is not properly initialized");
    adpDef = initDef;
    adpDef.createInput( ArgumentType.SINGLE_LINE, NAME, "Name to greet", null );
}
}
}
return adpDef;
}

    public static void main(String[] arg) {
        oracle.panama.core.util.Locator.getInstance();
        System.out.println("Start HelloAdapter");
        HelloAdapter ha = new HelloAdapter();
        Arguments args = oracle.panama.core.csc.CSCFactory.makeArguments();
        try {
            args.setInputValue( GREETING, "Hey" );
            AdapterDefinition ad = ha.getInitDefinition();
            System.out.println("InitDefinition:\n" + ad);
            ha.init(args);
            ad = ha.getAdapterDefinition();
            System.out.println("AdapterDefinition:\n" + ad);
            args.setInputValue( NAME, "Joe" );
            ServiceRequest sr =
oracle.panama.core.csc.CSCFactory.makeServiceRequest(null, args);
            Element result = ha.invoke(sr);
            System.out.println("Result:\n" + XMLUtil.valueOf(result));
        } catch (Exception ex) {
            System.out.println("Exception " + ex.getMessage());
            ex.printStackTrace();
        }
    }
}

```

```
        ((oracle.panama.PanamaException)ex).getDetail().printStackTrace();
    System.exit(-1);
    }
    System.out.println("Exit HelloAdapter");
System.exit(-1);
    }
}
```

When invoked, the adapter returns this result:

```
<SimpleResult>
  <SimpleText title="Oracle Portal-to-Go Server Hello Sample">
    <SimpleTextItem name="message" title="Portal-to-Go says:">
      <greeting> <name>!
    </SimpleTextItem>
  </SimpleText>
</SimpleResult>
```

The following sections describe each method in the sample adapter.

init()

The `init()` method initializes the adapter. Every adapter in Portal-to-Go must implement an `init()` method. Portal-to-Go calls the initialization method once, when the adapter is first instantiated.

The `init()` method takes an argument of type `Arguments`. `Arguments` is a convenience class for passing arguments. It includes various methods for setting and retrieving argument values. If invoked without an `Argument` object, `init()` throws an `AdapterException`. The sample `init()` method retrieves the initialization parameter value, the form of greeting, by calling `args.getInputValue()`. It passes the form of the greeting to the method, as the parameter `GREETING`. The default greeting set by the adapter is "Hello".

The contents of the initialization method must be synchronized to ensure that the class is not initialized again in another thread.

The sample sets the state variable, `initialized`, to true. Any adapter you create should similarly keep track of its own state. In a typical adapter implementation, the `init()` method would also contain logic for connecting to the content source.

invoke()

The `invoke()` method is the primary request handler. The master service calls this method every time a client makes a request. The adapter must be initialized before `invoke()` can be called.

The `invoke()` method takes an argument of type `ServiceRequest`. It returns an XML element that holds the result.

Upon receiving an end user request, the Portal-to-Go request manager creates a `ServiceRequest` object. The `ServiceRequest` contains any parameter values specified by the user, service alias, or master service. The `ServiceRequest` object also contains user information.

The sample `invoke()` builds a Simple Result document, using methods in the packages `org.w3c.dom.Element` and `org.w3c.dom.Text`. First, it creates the root result element:

```
Element result = XML.makeElement("SimpleResult");
```

Next, it creates a `SimpleText` element, sets its title attribute, and appends the element to the root element:

```
Element st = XML.makeElement("SimpleText");
st.setAttribute("title", "Oracle Panama Server HelloAdapter Sample");
result.appendChild(st);
```

As defined in the Simple Result DTD, a `SimpleTextItem` is a required child element of `SimpleText`. The sample `invoke()` method creates a `SimpleText` element, sets its title attribute, and appends the element to the root element:

```
Element sti = XML.makeElement("SimpleTextItem");
sti.setAttribute("name", "message");
sti.setAttribute("title", "Portal-to-Go says:");
st.appendChild(sti);
```

It then retrieves the input parameter value, appends it to the result document, and returns the result to the caller, the master service:

```
String name = sr.getArguments().getInputValue( NAME );
Text txt = XML.makeText( greeting + " " + name + "!" );
sti.appendChild(txt);
return result;
```

getInitDefinition()

Every adapter must implement `getInitDefinition()`. This method enables master services to determine the initialization parameters used by the adapter.

The `AdapterDefinition` class encapsulates adapter parameters. Your adapter should instantiate an `AdapterDefinition` object to hold the parameter definitions. In the `getInitDefinition()` method, you define the initialization parameters in the `AdapterDefinition` object and return the `AdapterDefinition`.

The method creates the initialization parameter with the following call:

```
initDef.createInit( Argument.SINGLE_LINE, GREETING, "Greeting phrase", null );
```

The `createInit()` method has the following signature:

```
public InitArgument createInit(int type,
                               String parameterName,
                               String comment,
                               String[] options);
```

The following table describes the method's arguments:

Argument	Description
type	Defines the GUI input element used for the parameter in the Service Designer. Possible values: SINGLE_LINE: for a single-lined text field. MULTI_LINE: for a larger, multi-lined text field. ENUM: for a combo box with a drop-down list of possible parameter values.
parameterName	The name of the parameter.
comment	An optional comment for the parameter.
options	If using an enumeration list for the parameter, you specify the possible values in this argument. Otherwise, set this to null.

getAdapterDefinition()

`getAdapterDefinition()` defines the adapter's runtime input parameters. You should create an `AdapterDefinition` object to hold the input parameters. You use the `AdapterDefinition.createInput()` method to create input parameter definitions. This method is similar to `createInit()`. It has the following signature:

```
public InputArgument createInput( int type,
                                String parameterName,
                                String comment,
                                String[] options);
```

See "[getInitDefinition\(\)](#)" for descriptions of each argument.

The `getInitDefinition()` method cannot be invoked on an uninitialized adapter. Therefore, the method first tests the adapter's state.

Managing Arguments

Portal-to-Go provides the following interfaces for managing adapter arguments:

- `InitArgument`
- `InputArgument`
- `OutputArgument`

The interfaces implement the `Arguments` interface. They include methods for creating, setting, and retrieving arguments. You can use the `OutputArgument` interface to filter content returned by an adapter and to link adapter output to service input.

Filtering Adapter Output

By filtering adapter output, you can apply a test to the content returned by an adapter. This enables you to deliver a service to an end user only if the results of the service (particularly a service invoked as a scheduled job) meet a specified condition.

For example, an end user may want to check the price of a certain stock and receive a notification if the stock reaches a certain price. You can use output filtering to create a test on the results of the scheduled adapter invocation, and deliver the service only if the result meets a condition the user sets.

Filtering Adapter Output Sample Code

The following code shows how to use output filters. The sample assumes that an output filter has been defined for a column in the database content source.

```
// Call provided API
// Get the arguments from the ServiceRequest
Arguments args = serviceRequest.getArguments();
```

```

// Assume that we have created our own SimpleResult wrapper
MySimpleResult simpleResult = new MySimpleResult();
MySimpleResult notFoundResult = MySimpleResult.NOT_FOUND_MESSAGE;
MyRow row;
boolean match = true;
int nRows = 0;
MyResult result = <get the content ....>;

while ((row = result.nextRow()) != null) {

    // Call provided API
    // only do this expensive op. if there is any filters defined.
    if (args.hasOutputFilters()) {
        match = true;
        for (int i = 0; i < row.getColumnCount(); i++) {
            // Call provided API
            OutputArgument farg =
                args.getOutputArgument(row.getColumnName(i));
            if (farg != null &&
                !farg.compare(row.getColumnValue(i))) { // if any filtering
                                                            // compare value.
                match = false;
                break;
            }
        }
    }
    if (match) { // only append row if it match the filtering.
        simpleResult.append(row);
        nRows++;
    }
}

// Call provided API
serviceRequest.setAnyResult(nRows != 0); // Let the ServiceRequest
// know if any result was fetched or not. This is used
// by the notification component to determine if the
// result should be sent or not.

return nRows != 0 ? simpleResult.getElement() :
notFoundResult.getElement();
...

```

Importing an Adapter into the Repository

After you create and compile the class that implements the adapter, you import the adapter into the Portal-to-Go repository.

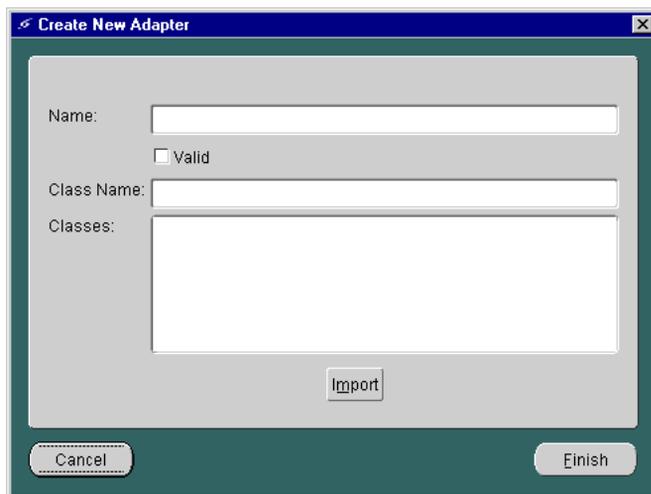
Note: You do not have to import an adapter into the repository if the adapter is in the CLASSPATH.

By importing the adapter into the repository, you make it available to master services. You use the Service Designer to import the adapter into the Portal-to-Go repository. Before importing the adapter, place the Java class in an archive, either a JAR or ZIP file, which you include in the system CLASSPATH.

You use the Create New Adapter form to install an adapter in the repository. To invoke the form:

1. Right-click Adapters in the Portal-to-Go Repository tree view.

2. Click Create New Adapter.



The form includes the following parameters:

Parameter	Value
Name	The name of the adapter. The name should be unique in the adapter folder.
Valid	Specifies whether the adapter is available to services. If selected, the adapter is available to services. If an adapter is invalid, the adapter is unavailable, and all master services that use the adapter are invalid.
Class Name	The Java class that either implements the adapter or serves as the entry point for the classes that implement the adapter.
Classes	The archived class or classes that implement the adapter. To import the archive, click Import and browse to the archive that implements the adapter.

Deleting an Adapter

You can delete an adapter as follows. If you delete an adapter used by active services, the Service Designer flags the services.

To delete an adapter:

1. Select the adapter in the Portal-to-Go repository tree view.
2. Right-click the adapter and click Delete.

Modifying Adapters

To modify an adapter:

1. Select the adapter in the Portal-to-Go repository tree view. The General tab of the properties panel appears in the right frame.
2. Modify the parameters and click Apply.

Extending the Stripper Adapter

The stripper adapter dynamically retrieves content from a given URL and processes the markup tags in the content. Unlike the WIDL adapter, the stripper adapter does not require you to map the source page to output and input bindings in a WIDL file. Instead, the stripper adapter retrieves and processes arbitrary content.

After retrieving content, the stripper adapter calls a Java class to process the markup tags in the content. The stripper adapter locates the processing classes, or strip levels, that are available to it in the **Strip.properties** file. This file also contains proxy settings, which you must set to use the strip adapter.

Portal-to-Go provides two strip levels:

Level	Description	Implementing Class
0	Retains all tags in the content.	Strip0
1	Strips all tags from the content.	Strip1

To customize markup processing, you can create your own strip levels. To create a strip level, you extend the `StripAbs` class. You then plug the strip level class into Portal-to-Go by referencing it in the **Strip.properties** file.

Naming the Strip Level Class

You must name your strip level class as follows:

Stripn.java

Where *n* is an integer representing the next available strip level. For example, given that Portal-to-Go provides strip levels 0 and 1, the first strip level you create should be level 2: **Strip2.java**.

Required Methods in the Strip Level Class

The strip level you create must implement the following methods:

- `public String exec (String in)`
- `public int getLevel()`

The `exec()` method should process the content or invoke the logic that processes the content retrieved by the adapter. The `getLevel()` should simply return the strip level.

Installing the Strip Level Class

Once you create and compile the strip level class, include it in the package `oracle.panama.adapter.stripper`.

Finally, include a reference to the strip level in the **Strip.properties** file. For example:

```
//New strip level 3  
stripper.strip3 = oracle.panama.adapter.stripper.Strip3
```

Transformers

This document describes how to create and manage Portal-to-Go transformers. Topics include:

- [What is a Portal-to-Go Transformer?](#)
- [Transformers Provided by Portal-to-Go](#)
- [Creating Transformers](#)
- [Java Transformers](#)
- [XSLT Stylesheets](#)
- [Testing the Transformer](#)
- [Managing Transformers with the Service Designer](#)

What is a Portal-to-Go Transformer?

Portal-to-Go transformers are Java programs or XSLT stylesheets that convert a Portal-to-Go document into either the target format or another Portal-to-Go format.

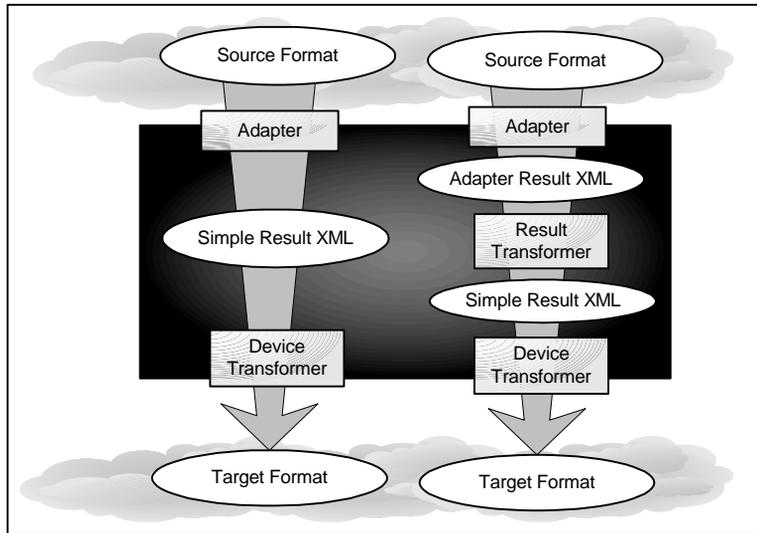
Generally, transformers encapsulate the logic for presenting content. Since presentation logic is often platform- and device-dependent, centralizing the presentation logic in transformers enables services to be portable across platforms and devices.

Portal-to-Go supports the following types of transformers:

- Result transformer
- Device transformer

Result transformers typically convert content from Adapter Result format to Simple Result format. The Adapter Result format is an intermediary format layer that

enables efficient exchange of user interface-independent data. You may use it, for example, to link chained services. A chained service is a Portal-to-Go service that invokes another service. An adapter can pass the service link using the Adapter Result format. A result transformer must render the Adapter Result document in Simple Result format before Portal-to-Go can deliver it to the user.



Device transformers convert Simple Result documents into the format of the target device. You can use two types of device transformers in Portal-to-Go:

- Default transformer
- Custom transformer

Portal-to-Go requires you to associate a transformer to each logical device. The transformer associated with a logical device is the device's default transformer.

Default transformers are typically generalized for a markup format, but they can also be specific to a target device.

Portal-to-Go uses the device's default transformer to convert any service targeted for that type of device, unless a custom device transformer overrides the default transformer. A custom device transformers enables you to control how a specific service appears on a specific device. While several logical devices can use a single default transformer, a custom transformer can be associated with only one master

service and one logical device. The custom transformer optimizes the presentation of that service for a particular device.

Note: A logical device is a repository object that represents either a physical device or an abstract device, such as an email server.

Transformers Provided by Portal-to-Go

Portal-to-Go provides transformers that convert Portal-to-Go XML for the following target formats:

Format	Description
HDML	The Handheld Devices Markup Language is a version of HTML designed specifically for mobile phones.
Plain text	Converts Simple Result format to plain text. This transformer is implemented as an XSLT.
Plain text – Java	Converts Simple Result format to plain text. This transformer is implemented as a Java class.
WML 1.1	The Wireless Markup Language standard defined by the WAP Forum.
Tiny HTML	The Tiny HTML format is a lightweight version of HTML, suitable for most PDAs. It lacks advanced HTML features, such as frame and JavaScript support.
VoxML	Motorola's markup language for voice-capable gateways.

Creating Transformers

The Portal-to-Go initial repository includes transformers for several target formats, including WML, HDML, and VoxML. By modifying the transformers provided with Portal-to-Go, or creating new ones, you can target new device platforms and optimize content presentation for specific devices.

Transformers not only map source tags to target format tags, they can manipulate content. They can rearrange, filter, and add text (such as boilerplate text). This enables you to present content in the format, as well as the form factor best suited for the target device.

When you create a transformer, you map the elements in the source content to the result format. For example, the Tiny HTML transformer, which is included in the initial Portal-to-Go repository, maps several Simple Result elements as follows:

Simple Result Source	HTML Result
<code><SimpleResult>Document</SimpleResult></code>	<code><html>Document</html></code>
<code><SimpleText title="Accounting"></code>	<code><h2>Accounting</h2></code>
<code><SimpleTextItem name="Employee">Scott</SimpleTextItem></code>	<code><p>Employee: Scott</p></code>

Similarly, when you create a transformer, you create a logical mapping between the abstract user interface elements represented by the Simple Result elements and the target format.

You can implement Portal-to-Go transformers as either Java transformers or XSLT stylesheets.

Java Transformers

To create a Java transformer for Portal-to-Go, you extend the `Transformer` interface. The class must be thread-safe and it must provide an empty constructor. The transformer should return a `String` object, which is the result document.

You can create Java transformers using either the DOM interface or the SAX interface. The DOM interface accesses and manipulates the compiled document object. The SAX interface interacts directly with the parsing process. The Oracle parser supplements these standards-based interfaces with extensions. For more information, see the *Portal-to-Go API Specification*.

The following section describes the components of a transformer class implementation.

Sample Java Device Transformer

Portal-to-Go includes a Java transformer that converts Simple Result documents to plain text. While the transformer does not create markup tags in the resulting document, it does apply simple text formatting elements, such as line breaks and tabs. Though simple, these elements illustrate how you can convert Simple Result elements into another format.

```
package oracle.panama.core.xform;  
import org.w3c.dom.NodeList;
```

```
import org.w3c.dom.Element;
import oracle.panama.PanamaException;
import oracle.panama.core.LogicalDevice;
import oracle.panama.core.Service;
import oracle.panama.Arguments;
import oracle.panama.core.parm.PanamaRequest;
import oracle.panama.core.parm.AbstractRequest;

public class SimpleResultToText implements Transform {
    public SimpleResultToText() {}

    private String format(Element el) {
        if (el == null) {
            return "";
        }
        StringBuffer buf = new StringBuffer();
        String name = el.getTagName();
        if (name != null && name.length() > 0) {
            buf.append(name);
            buf.append(": ");
        }
        buf.append(el.getNodeValue());
        return buf.toString();
    }

    public String transform(Element element, LogicalDevice device)
        throws PanamaException {
        PanamaRequest req = AbstractRequest.getCurrentRequest();
        Service service = req.getService();
        StringBuffer buf =
            new StringBuffer((service == null) ? "" : service.getName());
        NodeList list = element.getElementsByTagName("*");
        Element el;
        String tag;
        boolean newRow = false;
        for (int i = 0; i < list.getLength(); i++) {
            el = (Element)list.item(i);
            tag = el.getTagName();
            if (tag.equals("SimpleRow")) {
                newRow = true;
                buf.append("\n");
            } else if (tag.equals("SimpleCol")) {
                if (!newRow) {
                    buf.append("\t");
                } else {

```

```
        newRow = false;
    }
    buf.append(format(el));
} else if (tag.equals("SimpleText") ||
           tag.equals("SimpleForm") ||
           tag.equals("SimpleMenu")) {
    newRow = true;
    buf.append("\n");
} else if (tag.equals("SimpleTextItem") ||
           tag.equals("SimpleFormItem") ||
           tag.equals("SimpleMenuItem")) {
    if (!newRow) {
        buf.append("\n");
    } else {
        newRow = false;
    }
    buf.append(format(el));
}
}
return buf.toString();
}
}
```

The following sections describe each method in the transformer.

SimpleResultToText()

The `SimpleResultToText()` method is the default constructor for instantiating the transformer. Any transformer you create must also provide an empty default constructor.

format()

This transformer uses the `format()` method to format text, form, and menu items. The `format()` method uses the tag name as the item label, followed by the text of the item. `Element.getTagName()` returns the name of the element.

For example, given the following XML element:

```
<User>Scott</User>
```

The `format()` method returns:

```
User: Scott
```

transform()

The `transform()` method is the only required method in a transformer class. In the sample, the `transform()` method places each element from the source document in a node list. A node list is an indexed collection of elements.

```
NodeList list = element.getElementsByTagName("*");
```

The `transform()` method then steps through every element in the node list, testing each against the elements in the Simple Result DTD. If the source element matches the test Simple Result element, the method formats the element. It then appends the results to a buffer which, when finished, is returned to the caller.

XSLT Stylesheets

XSLT stylesheets are XML documents that specify the processing rules for other XML documents. The XSLT processor included with the Oracle XML processor conforms to the final W3C XSLT specification, Working Draft of August 13, 1999.

An XSLT stylesheet, like a Java transformer, is specific to a particular DTD. It should handle all elements declared in a DTD. When it finds the element in a source document, it follows the rules defined for the element to format its content.

XSLT stylesheets can include complex pattern matching and result handling logic. They typically include literal result elements, such as the target format markup tags.

Sample XSLT Device Transformer

The following transformer, which is included in the Portal-to-Go initial repository, is the XSLT version of the Java transformer described in "[Sample Java Device Transformer](#)". It converts Simple Result documents to plain text.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
  <xsl:template match="/">
    <xsl:apply-templates/></xsl:apply-templates>
  </xsl:template>
  <xsl:template match="SimpleTextItem | SimpleFormItem | SimpleMenuItem">
    <xsl:text></xsl:text>
    <xsl:value-of select="."/></xsl:value-of>
  </xsl:template>
  <xsl:template match="SimpleRow">
    <xsl:text></xsl:text>
    <xsl:for-each select="./SimpleCol">
      <xsl:text></xsl:text>
      <xsl:value-of select="."/></xsl:value-of>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

```
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

The sample illustrates the processing sequence for XSLT stylesheets. The stylesheet first matches, or selects, a `SimpleResult` element using pattern-matching semantics. The `<xsl:template match="/">` element, for example, matches the document's root element. It then uses the `apply-template` element to process the contents of a found element, including all sub-elements. A `select` attribute limits the processing scope to a specified sub-element.

The stylesheet then descends the source element tree, selecting and processing each sub-element. Character instructions, such as `value-of` and `for-each`, manipulate the content of matching elements. The `value-of` element extracts the actual content of the element. The `for-each` element applies iterative processing.

The sample does not include literal text elements. The following illustrates the use of literal text elements:

```
<xsl:template match="SimpleTextItem">
    <P>
        <xsl:value-of select="."/>
    </P>
</xsl:template>
```

This XSLT element maps the content of `SimpleTextItem` to HTML paragraph tags. For example, if passed the following `Portal-to-Go` element:

```
<SimpleTextItem>Scott</SimpleTextItem>
```

The XSLT segment would produce the following:

```
<P>
    Scott
</P>
```

Sample XSLT Result Transformer

The following result transformer segment shows how a result transformer can enable chain services. It uses the results of a user's selection to link to a target, which is the subsequent service in the chain.

```
<xsl:template match="AdapterResult">
    <SimpleResult>
        <SimpleMenu name="Query Results">
            <xsl:for-each select="Table/Row">
```

```

<xsl:if test="position() &#60; 8">
  <SimpleMenuItem>
    <xsl:attribute name="target">
      ___REQUEST_NAME___?PAservicepath=___SERVICE_URL_ENC___
      &#38;PAsession=Result&#38;href=<xsl:value-of
      select="./href"></xsl:value-of></xsl:attribute>
    <xsl:value-of select="./Title"></xsl:value-of>
  </SimpleMenuItem>
</xsl:if>
</xsl:for-each>
</SimpleMenu>
</SimpleResult>
</xsl:template>

```

The XML parser processes the document using the result transformer. Variables are preceded and terminated by underscore characters. The previous result transformer, for example, includes the variables `REQUEST_NAME` and `SERVICE_URL_ENC`.

`REQUEST_NAME` represents the name of the JSP file that executes the request. `SERVICE_URL_ENC` represents the path to a Portal-to-Go service in URL-encoded format. In the example, this is the path to a subsequent service in a service chain.

For example, given the following string:

```
http://www.panama.com/master/parm.jsp?PAservicepath=/master/getQuote
```

- `__REQUEST_NAME__` is "parm.jsp"
- `__SERVICE_NAME__` is "getQuote"
- `__SERVICE_URL__` is "/master/getQuote"

You can select the values of these variables in a transformer directly with a construct such as the following:

```
<xsl:value-of select="//_REQUEST_NAME"/>
```

Testing the Transformer

Portal-to-Go provides a command-line utility you can use to test the transformers you create. The `Xslt` utility takes a source document and an XSLT stylesheet, and writes the result document to standard output.

You invoke `Xslt` from the command prompt as follows:

```
java oracle.panama.util.Xslt mystylesheet.xml < myxml.xml
```

Managing Transformers with the Service Designer

The Service Designer enables you to manage transformers in the Portal-to-Go repository. Using the Service Designer, you can create and update transformers, and remove them from the repository.

Creating a Transformer in the Repository

You use the Create New Transformer form in the Service Designer to create a transformer in the repository. To display the form:

1. Right-click Transformers in the Portal-to-Go repository tree view.
2. Click Create New Transformer to display the first form in the sequence.

The Create New Transformer form appears.

The form includes the following parameters:

Parameter	Value
Name	The transformer name. This must be a unique name in the transformer folder of the repository tree.

Parameter	Value
Master Service	In the case of a custom transformer, the master service associated with the transformer.
Logical Device	In the case of a custom transformer, the logical device associated with the transformer.
Java Transformer	Specifies a Java class transformer implementation.
Class Name	The name of the class that implements the transformer.
XSL Transformer	Specifies an XSLT stylesheet transformer implementation.
Style-sheet	The actual XSLT stylesheet that implements the transformer. You can cut and paste a transformer from another editing environment into this field.

Modifying a Transformer

To modify a transformer in the Portal-to-Go repository, follow these steps:

1. Expand the Transformers folder in the Portal-to-Go repository tree view.
2. Click the transformer you want to modify. The General tab of the properties panel appears.

Modify the parameters as required and click Apply. For information on the parameters in the transformer properties panel, see "[Creating Transformers](#)".

Removing a Transformer from the Repository

To remove a transformer from the repository:

1. Expand the Transformers folder in the Portal-to-Go repository tree view.
2. Right-click the transformer you want to delete.
3. Click Delete. The Confirm Delete dialog appears.
4. Click Yes to confirm the action.

Logical Devices

This document explains how to create and manage logical devices. Topics include:

- [Overview](#)
- [Creating a Logical Device](#)
- [Modifying a Logical Device](#)
- [Deleting a Logical Device](#)
- [Determining the Client Device Type](#)

Overview

A logical device is a repository object that represents either a physical device, such as a Nokia mobile phone, or an abstract device, such as email. Logical devices represent the interface between Portal-to-Go transformers and the target devices or applications.

Portal-to-Go determines the type of device that is requesting a service using the user agent parameter of the HTTP request. When adding support for a new device type, you generally need to:

1. Create the logical device in the repository.
2. Create a transformer for the device.
3. Configure Portal-to-Go to recognize the new device.

In many cases, when adding support for a new device type, you can simply modify or re-use an existing transformer or logical device.

Creating a Logical Device

You use the Logical Devices form sequence in the Portal-to-Go Service Designer to create a logical device in the repository.

To display the form sequence:

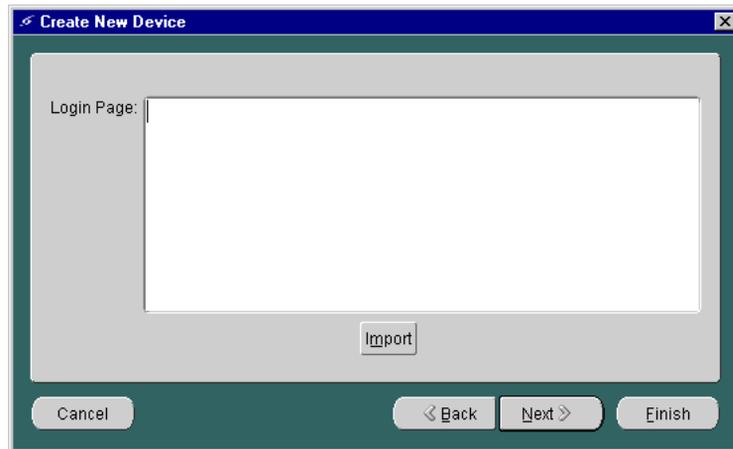
1. Right-click Logical Devices in the Portal-to-Go repository tree view.
2. Click Create New Device to display the first form in the sequence.

The first form in the Create New Device sequence has the following parameters:

Parameter	Value
Name	The logical device name. This must be a unique name.
MIME Type	The logical device MIME type.
Encoding	Content encoding for the logical device. Content encoding is used to transport the result of the device type.

Parameter	Value
Needs URL Caching	When selected, the URL-encoding request parameters are cached on the server and only small pointers are encoded within the URL that is sent to the logical device. Use this option for devices with limited memory. You should also select this option if the browser does not support cookies.
Default Transformer	Specifies the transformer to use to convert content for the device.
Prolog	Specifies the format of the prolog required by the device. The prolog frequently includes processing instructions and meta tags.

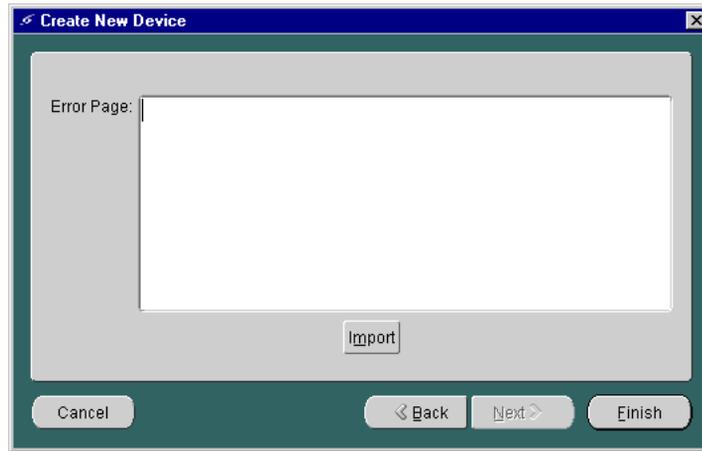
The second form in the Create New Device sequence appears as follows:



The second form in the Create New Device sequence includes the following parameter:

Parameter	Value
Login Page	The XSLT stylesheet that generates the login page for the device. Click Import and choose an XSLT stylesheet from the Open dialog.

The third form in the Create New Device sequence appears as follows:



The third form in the Create New Device sequence has the following parameter:

Parameter	Value
Error Page	The XSLT stylesheet that generates the error page for the device. Click Import and choose an XSLT stylesheet from the Open dialog.

Click Finish to create the object in the repository.

Modifying a Logical Device

The Portal-to-Go Service Designer enables you to modify a logical device in the repository. To modify a logical device:

1. Expand the Logical Device folder in the Portal-to-Go repository tree view.
2. Click the device you want to modify. The Edit Logical Device form appears in the right frame.
3. Modify the desired parameter(s). You can toggle between the General, Login Page, and Error Page tabs.
4. Click Apply.

Deleting a Logical Device

To delete a logical device from the repository:

1. Expand the Logical Device folder in the Portal-to-Go repository tree view.
2. Right-click the device you want to delete.
3. Click Delete.
4. Click Yes in the Confirm Delete dialog to confirm the action.

Determining the Client Device Type

When Portal-to-Go receives a service request, it determines the device type that submitted the request by querying the user agent parameter of the HTTP header. The **System.properties** file specifies the Portal-to-Go class that queries the header and makes the determination:

```
locator.useragent.class=oracle.panama.core.xform.UserAgentImpl
```

The `UserAgentImpl` class is a generic implementation of the `UserAgent` interface. The `UserAgent` interface takes the HTTP user agent parameter and returns the device type as a string corresponding to the device name in the repository. If the device type is unknown, it returns NULL.

You can create your own implementation of the `UserAgent` interface to implement specialized logic for determining the client device type. For example, if the HTTP gateway that passes requests to Portal-to-Go includes the telephone number of the client, you can implement a user agent that uses the telephone number to retrieve end user information from a directory.

Once you implement your own user agent, you must point the user agent class locator in the **System.properties** file to your implementation:

```
locator.useragent.class=oracle.panama.core.xform.MyUserAgentImpl
```

Portal-to-Go uses a properties file, **useragent.properties**, to map the HTTP user agent parameter to a logical device in the Portal-to-Go repository. The `UserAgentImpl` class retrieves this information from **useragent.properties** dynamically. When you add logical devices to the repository, you should map the device to the corresponding HTTP user agent parameter in the **useragent.properties** file.

To determine what information Portal-to-Go actually receives in the HTTP header, you can check the Portal-to-Go system log files. For full request manager logging,

set the request manager debug flag. Using the probe monitor, click Debug, then click “Change debug flag”.

Note: The **useragent.properties** file also specifies the default logical device that Portal-to-Go uses if it cannot determine the device type. By default, this is set to WML11.

The **useragent.properties** file uses the following syntax to map an HTTP user agent parameter to a Portal-to-Go logical device:

```
[ID].useragent=[UserAgent startstring from HTTP Request]
[ID].logicaldevice=[which logical device to use]
```

The logical device should appear on the line immediately following the corresponding user agent string. They should use the same ID, which is simply a unique number in the **useragent.properties** file. The ID associates a user agent string in the request with a logical device. For example:

```
30.useragent=MOTO
30.logicaldevice=Motorola
```

The `UserAgentImpl` class matches the actual header string to the longest matching user agent mapping in the **useragent.properties** file. That is, given the following mappings:

```
42.useragent=UP.Browser/3.01-IG01
42.logicaldevice=HDML

43.useragent=UP.Browser
43.logicaldevice=WML11
```

The `UserAgentImpl` class would match the header string “UP.Browser/3.01” to the HDML logical device, rather than the WML11 logical device.

Once you create or modify a device mapping, you can restart Portal-to-Go or force a re-read of the property file using the probe runtime monitor (on port 8090), so that the changes to the property file take effect.

Integrating Portal-to-Go

This document discusses the features that enable you to integrate Portal-to-Go with other systems. It provides information on the listening agents and servers Portal-to-Go uses to enable email and Short Message Service (SMS) support. Topics include:

- [Asynchronous Notifications](#)
- [Notification Engine](#)
- [SMS Listener](#)
- [Probe Runtime Monitor and Management](#)
- [RMI Server](#)
- [FTP Server](#)

Asynchronous Notifications

Asynchronous notifications are scheduled requests that Portal-to-Go delivers to end users. The end user configures notifications from the Personalization Portal. The Portal-to-Go system manages asynchronous notifications using the Oracle8i Advanced Queuing package. Portal-to-Go uses asynchronous notifications to perform requests and notifications.

Request Queue

Portal-to-Go processes requests as follows:

1. A job is created or updated with a start date and a specified interval. An agent is connected to the job. The agent specifies the address and device type to which the result is sent.

2. If the start date is greater than the current date, the job is placed in the job scheduler.
3. When the scheduler executes the job, the Job object is activated and invoked. The result is placed in the notification queue. The Job object calculates the new execution date and saves it to the database. Once saved, the new execution date is added to the job scheduler with the new execution time.
4. If the job fails, it is not added to the job scheduler, and can only be activated by updating the start date again.

Notification Queue

Portal-to-Go processes notifications as follows:

1. The notification queue receives a message, such as an SMS or email message.
2. The notification queue immediately dispatches the message to the notification listener.

Component Requirements

Asynchronous messaging requires the following:

- The repository must be stored in Oracle8i.
- The database JOB option must be installed in Oracle8i and activated. See the **init.ora** file configuration in the Oracle8i documentation.
- The database AQ option must be installed in Oracle8i and activated. See the **init.ora** file configuration in your Oracle8i documentation.

Notification Engine

The notification engine implements the interface `oracle.panama.util.NotificationListener`, and is registered in the system properties key `locator.notification.dispatcher.class`.

The current implementation of the notification engine is the class:

```
oracle.panama.util.NotificationDispatcherImpl
```

This class dispatches messages to SMS or email agents using the following helper classes:

- `oracle.panama.util.sms.SmsSender`
- `oracle.panama.util.mail.MailSender`

You can customize all notification engine properties in the **Notification.properties** file, which you can find in the oracle/panama/core/admin directory. This property file contains the following parameters:

Key	Type	Modifiable?	Description
sms.account.id	AccountId	Y	The large account number of this system.
sms.server.name	HostName	Y	The SMS TCP server host name.
sms.server.port	Integer	Y	The SMS TCP server listening port.
dispatcher.logicaldevice. email.name	String	Y	The name of the email logical device in the repository.
dispatcher.logicaldevice. sms.name	String	Y	The name of the SMS logical device in the repository.
mail.server.name	HostName	Y	The SMTP server host name. This is mandatory.
mail.from	String	Y	The "from" address in mail headers. This is mandatory.
mail.organization	String	Y	The organization name in mail headers. This is optional.
mail.domain	DomainName	Y	The network domain name to send to the SMTP service when connecting. This is only needed if the SMTP server requires a HELO greeting.
mail.subject	String	Y	The default subject of mail messages.

Component Requirements

The current notification implementation requires the following components:

- An SMS-C TCP server available on the network.
- An SMTP service available on the network.

Limitations

The current notification engine has the following limitations.

- It can only construct SMS, Universal Computer Protocol, UCP messages of operation 01.
- The current character encoding is based on ISO-8859-1 for both SMS and email messages.

SMS Listener

The SMS Listener processes SMS requests and returns results as SMS messages. It processes requests by performing the following:

1. Authenticates the user by finding an agent with the actual address (phone number).
2. Selects the agent by searching the order of logical devices specified in the **Notification.properties** file. These include SMS and TTML devices. The message appears as a single line with the relative path of the service name. The path is relative to the users home/root folder.
3. Invokes the service.
4. Sends the result back to the caller by placing the message in the notification queue.

The SMS Listener uses notification functionality and shares properties configured in the **Notification.properties** file in the oracle/panama/core/admin directory.

This properties file has the following parameters:

Key	Type	Modifiable?	Description
sms.listener.port	Integer	Y	The TCP port on which the daemon listens.
sms.listener.logicaldevice.search.list	StringList	Y	The logical device search order, with highest priority first. The name must be the same as in the repository.

The implementation is managed by the daemon:

```
oracle.panama.util.sms.SmsServer
```

Component Requirements

The SMS listener contains the following component requirements:

- The notification components, both the queue and the engine, must be up and running.
- The caller must have a valid agent which specifies a telephone number, and a device that matches a device in the property file.

Probe Runtime Monitor and Management

The probe is a daemon that displays runtime information about:

- Log files
- Active objects
- Active sessions
- Active threads

The probe performs simple runtime management including:

- Shutting down an active object instance
- Refreshing persistent attributes on an active object
- Setting the global debug flag to on or off
- Reloading properties for device recognition when adding a new device to a running system.

By default, the Portal-to-Go installation process installs and initiates the probe runtime monitor with the listening port 8090. To access the probe, from a browser enter the following URL:

`http://host_name:8090`

You can configure the probe component by editing the **www-server.properties** file in the `oracle/panama/core/probe` directory.

The **www-server.properties** file has the following parameters:

Key	Type	Modifiable?	Description
port	Integer	Y	The TCP listening port.
root	Path	Y	The root directory for log files.
timeout	Milliseconds	Y	Timeout for a request TCP socket.

Key	Type	Modifiable?	Description
workers	Integer	Y	The number of working request threads to start up.
log	String	Y	True if the activity should be logged, otherwise false.

The following daemon manages the implementation:

```
oracle.panama.core.probe.WebServer
```

RMI Server

You can manage objects in the Portal-to-Go repository from a remote client using the RMI (Remote Method Invocation) server. The RMI server is a daemon that enables you to access and manipulate Portal-to-Go objects using an RMI interface. Currently, Portal-to-Go automatically generates the RMI component from the interface definitions using introspection of the compiled Java interfaces. See `oracle.panama.core.util.GenerateRMI` for more information.

An RMI session is stateless and each update call is always completed with a commit or rollback. For this reason, you can manage all persistent objects from a remote client. Management actions take immediate effect when the RMI server works at the object level of the system. To connect to an RMI daemon, you must provide a valid administrator user name and password.

You can configure the RMI component by editing the **Rmi.properties** file in the `oracle/panama/core/admin` directory.

The **Rmi.properties** file has the following parameters:

Key	Type	Modifiable?	Description
rmi.server.port	Integer	Y	The TCP port on which the daemon listens.
rmi.server.name	String	Y	The name of the RMI server.

The implementation is managed by the daemon:

```
oracle.panama.core.rmi.server.ServerImpl
```

FTP Server

The FTP server is a daemon that publishes the repository as an FTP file system. The repository folder is represented as an FTP directory, and repository objects are represented as FTP files. The FTP server makes it possible to upload and download users, services, and other repository objects as XML documents.

FTP support allows you to:

- Authenticate users (only administrators can use this utility).
- List directories and files.
- Download XML-enabled persistent objects.
- Upload XML-enabled persistent objects.
- Create subdirectories (service tree folders only).
- Delete directories (service tree folders only).
- Delete any object.
- Rename directories and files.

Note: When uploading new objects, you do not need to assign values to the unique name or `objectId_` attributes. The upload feature uses "update or create" logic. If either the unique name or `objectId_` attribute is set to the same value as an existing object, then the existing object is updated. The unique name of a service is the `url` attribute. For all other objects, the unique name is the `name` attribute.

The FTP component can be configured by editing the **Ftp.properties** file in the `oracle/panama/core/admin` directory.

The **Ftp.properties** file has the following parameter:

Key	Type	Customizable	Description
<code>ftp.server.port</code>	Integer	Y	The TCP listening port

Portal-to-Go Extensibility

This document discusses the components you can use to extend Portal-to-Go functionality. Topics include:

- [Overview](#)
- [Daemons](#)
- [Hooks](#)
- [Extending the Core Components](#)

Overview

Portal-to-Go supports system-wide extensibility. Not only can you create adapters and transformers that extend its core functionality, you can modify the internal processing of its core objects and create specialized components for system monitoring and integration.

Portal-to-Go uses the **System.properties** file to determine at runtime which components to use for system processes. When you create a class to implement specialized functionality, you plug the class into the Portal-to-Go architecture by identifying the class as a key value in the **System.properties** file.

Portal-to-Go supports the following types of plug-in classes:

- **Daemons:** Daemons are processing objects that execute in their own thread. Typically, they listen for and respond to events.
- **Hooks:** Hooks process objects that execute in the same thread as Portal-to-Go.

To customize or implement a new component in the Portal-to-Go architecture:

1. Create the class that implements the component customization or extension.

2. Reference the class in the **System.properties** file.
3. Configure any parameters that are specific to the component in the appropriate properties files.

You must implement the pluggable class as a singleton (only a single instance of the class can exist in the system at a time), with static class methods. You should include the class in the system CLASSPATH, so that it can be loaded by the standard Java Virtual Machine class loader. When changing a component, you must shut down and restart the Portal-to-Go server processes.

Daemons

You create your own Portal-to-Go system daemons by implementing the following interfaces:

- `oracle.panama.util.NotificationListener`
- `oracle.panama.core.admin.LogListener`

Notification Listeners

You use the notification listener interface to implement your own Portal-to-Go notification engine. The notification engine responds to incoming messages. Currently, the default implementation of the notification engine handles email (SMTP) and Short Message Service (SMS) messages.

You can use only one notification engine in a Portal-to-Go system.

You specify domain-specific runtime parameters for the notification engine in the **Notification.properties** file. This file contains settings for the SMS listener port and the server name.

Log Listeners

You implement the log listeners interface to create classes that respond to log events.

You can create any number of log listeners in Portal-to-Go.

Hooks

Portal-to-Go provides the following types of hooks:

Name	Method	Purpose
Service hook	<code>MasterServiceHook.isVisible(service, request)</code>	Determines whether a service is visible (such as location-dependent services).
Request hook	<code>ParmHook.requestBegin(request)</code>	Executes when a request arrives.
User authentication hooks	<code>UserAuthenticationHook.checkUser(name)</code> <code>UserAuthenticationHook.authenticate(info, request)</code>	Gets the user from an external source. Authenticates the user.
Provisioning hook	<code>ProvisioningHook.getRootFolder(user)</code>	Creates a root folder for a new user.
Device identification hook	<code>UserAgent.findDeviceType(request)</code>	Identifies the user's device.
Notification listener	<code>NotificationListener.onMessage(NotificationEvent ev)</code>	Implements a new notification engine.

Extending the Core Components

In some cases, you may want to alter the internal processing for the Portal-to-Go objects. You can accomplish this using programmatic hooks provided by Portal-to-Go.

While you can extend internal object processing, you cannot alter the fundamental object model. The following sections provide a sample implementation that extends the master service. For more information regarding the hooks in object functionality, see the Portal-to-Go API specification.

Step 1: Override the Existing Implementation

You must be able to construct each persistent object with either a known object identifier or an empty constructor, as follows:

```
public class MyMasterService extends MasterService {
    public MyMasterService() {
        super();
    }
}
```

```

public MyMasterService(Long id) {
    super(id);
}

// override some logic.
public void invoke(ServiceRequest request) throws Exception {
    super.invoke(request);
    sendMessageToOnlinePaymentSystem(request);
}
}

```

Step 2: Plug in the Implementation with a Locator

Each persistent object has two methods that the persistent locator must implement. The persistent locator must also implement the singleton pattern, as follows:

```

MasterService lookupMasterService(Long id);
MasterService newMasterService();

public class MyPersistentLocator extends PersistentLocatorImpl {
    private static PersistentLocator instance = null;

    // Always calls to the empty constructor of the superclass
    protected MyPersistentLocator() {
        super();
    }

    // The thread-safe singleton getter
    public static PersistentLocator getInstance() {
        if (instance == null) {
            MyPersistentLocator impl = null;
            synchronized (MyPersistentLocator.class) {
                if (instance == null) {
                    impl = new MyPersistentLocator();
                    instance = impl;
                }
            }
        }
        // do the initialization here needed for some components,
        // to avoid recursive loops
        // watch out for initialize() methods
        if (impl != null) {
            impl.initialize();
        }
    }
    return instance;
}

```

```
}

// Override newMasterService
public MasterService newMasterService() {
    return new MyMasterServiceImpl();
}

// Override lookupMasterService
public MasterService lookupMasterService(Long id) {
    MasterService m = (MasterService)getMasterSet().get(id); // always
                                                             // check the cache first

    if (m == null) { // no hit
        m = new MyMasterServiceImpl(id); // activate an existing
                                         // object from storage
    }
    return m;
}
}
```

Step 3: Reference the Class in the System Properties File

Reference the new locator in the **System.properties** file:

```
locator.persistent.class=MyPersistentLocator
```

Simple Result DTD Notes

The Simple Result DTD is the Portal-to-Go internal representation of the result returned by an adapter. If an adapter does not return the result in this format, the master service must use the result transformer to convert the result into the Simple Result format.

The following notes describe the content of the DTD including the elements, the usage, the attributes, the children, and the DTD declarations.

SimpleResult

Element

SimpleResult is the result element.

Usage

This element contains the result.

Children

[SimpleContainer](#)

Attributes

title

DTD Declaration

```
<!ELEMENT SimpleResult (SimpleContainer+)>
```

```
<!ATTLIST SimpleResult
```

```
title CDATA #IMPLIED
```

>

SimpleContainer

Element

SimpleContainer is a compound UI consisting of several simple UI elements.

Usage

Used as a logical container for one or more simple elements.

Children

[SimpleText](#)

[SimpleMenu](#)

[SimpleForm](#)

[SimpleTable](#)

[SimpleImage](#)

[SimpleBreak](#)

[SimpleHref](#)

[SimplePhone](#)

[SimpleEmail](#)

[SimpleHelp](#)

DTD Declaration

```
<!ELEMENT SimpleContainer (SimpleHelp?, (SimpleText | SimpleMenu
    | SimpleForm | SimpleTable
    | SimpleImage | SimpleBreak | SimpleHref | SimplePhone |
    SimpleEmail)+)>
<!ATTLIST SimpleContainer
    title CDATA #IMPLIED
>
```

SimpleText

Element

SimpleText is used for displaying one or more blocks of text.

Usage

Used for plain text.

Children

[SimpleTextItem](#)

[SimpleHelp](#)

Attributes

title

DTD Declaration

```
<!ELEMENT SimpleText (SimpleHelp?, SimpleTextItem+)>
```

```
<!ATTLIST SimpleText>
```

SimpleTextItem

Element

SimpleTextItem is a block of text.

Usage

Holds one block of text, normally a single paragraph.

Children

#PCDATA (the actual text)

Comments

There is no line feed at the end of each SimpleTextItem. See "[SimpleBreak](#)".

Attributes

name

title

DTD Declaration

```
<!ELEMENT SimpleTextItem (#PCDATA)>
<!ATTLIST SimpleTextItem
  name CDATA #IMPLIED
  title CDATA #IMPLIED
  >
```

SimpleMenu

Element

SimpleMenu displays a simple menu.

Usage

A single menu with selectable links.

Children

[SimpleMenuItem](#)

[SimpleHelp](#)

Comments

A SimpleBreak should be inserted wherever a linefeed is desired.

DTD Declaration

```
<!ELEMENT SimpleMenu (SimpleHelp?, SimpleMenuItem+)>
<!ATTLIST SimpleMenu>
```

SimpleMenuItem

Element

SimpleMenuItem is a single, selectable option in a [SimpleMenu](#).

Usage

This is a menu item that links to something.

Children

[SimpleMenu](#)

Attributes

target (the link target for this item)

icon (the name of the icon without an extension. The file extension is appended by the device transformer.)

DTD Declaration

```
<!ELEMENT SimpleMenuItem (SimpleMenu?)>
<!ATTLIST SimpleMenuItem
  target CDATA #REQUIRED
  icon CDATA #IMPLIED
  >
```

SimpleForm

Element

SimpleForm displays one or more input fields.

Usage

Used as a data-entry form.

Children

[SimpleFormItem](#)

[SimpleHelp](#)

Attributes

target (the link target for this form)

submit (the text for the submit button, default is "Submit")

DTD Declaration

```
<!ELEMENT SimpleForm (SimpleHelp?, (SimpleFormItem |  
SimpleFormSelect)+)>  
!ATTLIST SimpleForm  
target CDATA #REQUIRED  
submit CDATA #IMPLIED  
>
```

SimpleFormItem

Element

SimpleFormItem is a single input item in a simple form.

Usage

Used for user input.

Children

#PCDATA (a default value from the server; it deprecates the default attribute)

Attributes

value (see the description for "default")

default (a default value for optional fields, it is deprecated by the value of the element. See the child element for more information.)

name (the name of the variable)

title (specifies a title for the element, which may be used in the presentation of this object)

mandatory (indicates that the form item is mandatory)

maxLength (provides a maximum input length)

size (the width of the input field, if the input field is larger than the device screen, it is displayed in multiple lines)

format (the format of inputs, see the WML 1.1 specification for more information)

type (whether the input field should have "password" style, valid values are *text* or *password*)

DTD Declaration

```
<!ELEMENT SimpleFormItem (#PCDATA)>
<!ATTLIST SimpleFormItem
  name CDATA #REQUIRED
  title CDATA #IMPLIED
  value CDATA #IMPLIED
  default CDATA #IMPLIED
  mandatory (yes | no) "no"
  maxLength CDATA #IMPLIED
  size CDATA #IMPLIED
  format CDATA #IMPLIED
  type (text | password) "text"
>
```

SimpleFormSelect

Element

SimpleFormSelect is a selectable option menu in a simple form.

Usage

Presents a selectable number of options to the user. This is a radio box group.

Children

[SimpleFormOption](#)

[SimpleHelp](#)

Attributes

name (the name of the variable)

title (the display name for the variable)

DTD Declaration

```
<!ELEMENT SimpleFormSelect (SimpleHelp?, SimpleFormOption+)>
<!ATTLIST SimpleFormSelect
  name CDATA #REQUIRED
  title CDATA #IMPLIED
>
```

SimpleFormOption

Element

SimpleFormOption is a single option in a simple form.

Usage

A single option.

Children

#PCDATA (the text of the option)

Attributes

value (the value assigned when this option is selected)

DTD Declaration

```
<!ELEMENT SimpleFormOption (#PCDATA)>
<!ATTLIST SimpleFormOption
  value CDATA #REQUIRED
>
```

SimpleTable

Element

SimpleTable is for displaying tabular data.

Usage

Used for simple tabular data only.

Children

[SimpleTableHeader](#)

[SimpleTableBody](#)

Attributes

title (the title for the table)

DTD Declaration

```
<!ELEMENT SimpleTable (SimpleTableHeader?, SimpleTableBody)>
```

```
<!ATTLIST SimpleTable
```

```
title CDATA #IMPLIED
```

```
>
```

SimpleTableHeader

Element

SimpleTableHeader contains the table header.

Usage

Contains table header information, such as column headers.

Children

[SimpleCol](#)

Attributes

none

DTD Declaration

```
<!ELEMENT SimpleTableHeader (SimpleCol+)>
```

```
<!ATTLIST SimpleTableHeader>
```

SimpleTableBody

Element

SimpleTableBody contains the actual tabular data.

Usage

Contains table data rows.

Children

[SimpleRow](#)

Attributes

none

DTD Declaration

```
<!ELEMENT SimpleTableBody (SimpleRow+)>
```

```
<!ATTLIST SimpleTableBody>
```

SimpleRow

Element

SimpleRow contains a single row of data.

Usage

Stores a table row.

Children

[SimpleCol](#)

Attributes

none

DTD Declaration

```
<!ELEMENT SimpleRow (SimpleCol+)>
```

```
<!ATTLIST SimpleRow>
```

SimpleCol

Element

SimpleCol contains a single table cell.

Usage

Stores a single value for a table cell.

Children

#PCDATA (the column value)

DTD Declaration

```
<!ELEMENT SimpleCol (#PCDATA)>
```

```
<!ATTLIST SimpleCol>
```

SimpleImage

Element

SimpleImage displays an image.

Usage

Used to reference an external image.

Children

none

Attributes

target (the URL for this image, without an extension)

alt (the alternative text, if the device does not support images)

width (the width of the image)

height (the height of the image)

DTD Declaration

```
<!ELEMENT SimpleImage EMPTY>
```

```
<!ATTLIST SimpleImage
```

```
target CDATA #REQUIRED
alt CDATA #IMPLIED
width CDATA #IMPLIED
height CDATA #IMPLIED
>
```

SimpleBreak

Element

SimpleBreak provides a logical break in the layout.

Usage

Used to provide a logical break.

Children

none

Attributes

none

DTD Declaration

```
<!ELEMENT SimpleBreak EMPTY>
```

```
<!ATTLIST SimpleBreak>
```

SimplePhone

Element

SimplePhone references a phone number.

Usage

A phone number.

Children

#PCDATA (the name of the phone)

Attributes

target (a phone number, for example, +46705104725)

DTD Declaration

```
<!ELEMENT SimplePhone (#PCDATA)>
```

```
<!ATTLIST SimplePhone>
```

SimpleEmail

Element

SimpleEmail references an email address.

Usage

Used to reference an email address.

Children

#PCDATA (the name of the user)

Attributes

target (the email address)

DTD Declaration

```
<!ELEMENT SimpleEmail (#PCDATA)>
```

```
<!ATTLIST SimpleEmail
```

```
target CDATA #REQUIRED
```

```
>
```

SimpleHref

Element

SimpleHref provides an anchor to another resource.

Usage

An anchor to another resource.

Children

#PCDATA (the name of the link)

Attributes

target (the link)

DTD Declaration

```
<!ELEMENT SimpleHref (#PCDATA)>  
<!ATTLIST SimpleHref  
  target CDATA #REQUIRED  
>
```

SimpleHelp

Element

SimpleHelp indicates a help text.

Usage

Used for help text.

Children

#PCDATA (the name of the help text)

Attributes

none

DTD Declaration

```
<!ELEMENT SimpleHelp (#PCDATA)>  
<!ATTLIST SimpleHelp>
```

Glossary

adapter

A dynamically loaded Java class that acquires content from an external source, such as a Web site or a database, and converts the content into Portal-to-Go XML. Pre-built adapters include the Web Integration adapter, SQL adapter, and Strip adapter.

Adapter Result format

A general, user interface-independent content format. Content in Adapter Result format requires conversion to Simple Result format before it can be converted to the final target format.

bookmark

A link from a service to an external, device-compatible data source that does not require Portal-to-Go processing.

core

The Portal-to-Go component that manages the Portal-to-Go repository and service requests.

device portal

The interface where mobile device users access their Portal-to-Go services.

device transformer

A transformer that converts content from Simple Result format into the target format.

DOM Interface

Document Object Model. The interface that allows programs and scripts to access and transform processed XML documents.

DTD

Document Type Declaration. A file that defines the format elements for a type of XML document.

end user

A person who accesses a Portal-to-Go service from a client device.

filtering

The process of transforming content by replacing existing markup tags with tags that represent another format.

HDML

Handheld Device Markup Language. A reduced version of HTML designed to enable wireless pagers, cellular phones, and other handheld devices to access Web page content.

HTML

HyperText Markup Language. The document format that defines the page layout, fonts, and graphic elements, as well as the hypertext links to other documents on the Web.

JSP

JavaServer Pages. A technology based on Java servlets which separates the functions of Web page layout and content generation. JavaServer Pages technology enables the creation of server-generated Web pages incorporating dynamic content.

logical device

An object that describes either a physical device, such as a cellular phone, or an application, such as email. There is a default device transformer for each logical device.

master service

The core implementation of a service. The master service object invokes a specific adapter, and identifies the transformer used to convert content for the target device.

Personalization Portal

A Web-based interface that end users access to select services and configure their device portal. Users access the Personalization Portal from their desktop computers.

Portal-to-Go XML

A set of DTDs and XML document conventions used by Portal-to-Go to define content and internal objects.

private service tree

A service tree that is owned by a specific end user.

provisioning adapter

The adapter used to create, modify, and delete user objects in the Portal-to-Go repository.

public service tree

A service tree that is owned by the Portal-to-Go system, and can be accessed by any end-user.

repository

An Oracle8i database which stores all Portal-to-Go objects, such as users, groups, adapters, and services.

request

A query to initiate a desired Portal-to-Go service. Requests are submitted on behalf of end-users to the Portal-to-Go server.

request manager

The Portal-to-Go component that processes requests for services. The request manager authenticates the user, submits the request to the Portal-to-Go core, and retrieves the device type and any presentation settings. The request manager also forwards converted content from the transformer to the user.

request object

An XML document representing a request for service.

result transformer

A transformer that converts content from Adapter Result format into Simple Result format.

RMI

Remote Method Invocation. A standard for creating and calling remote objects. RMI allows Java components stored in a network to be run remotely.

sample repository

The initial Portal-to-Go repository, which includes pre-built objects such as transformers, adapters, and logical devices.

service

A core object used in a Portal-to-Go server to represent a unit of information requested by, and delivered to, a Portal-to-Go client. An end user typically sees a service as a menu item on a device or as a link on a Web page.

service alias

A pointer to a master service. When a service alias is placed in a service tree, the corresponding service becomes available to the owner or owners of the service tree.

service designer

The visual interface for creating and managing Portal-to-Go users, user groups, adapters, transformers, and services.

service tree

A tree data structure containing one or more services. Service trees make services available to end users.

Simple Result format

A content format that contains abstract user interface elements such as text items, menus, forms, and tables.

source format

The original format of content retrieved from an external data source by a Portal-to-Go adapter. For example, the source format of Web page content is HTML.

Strip adapter

An adapter that retrieves and adapts Web content dynamically.

strip level

The class used by the strip adapter to process markup tags in source content.

SQL adapter

An adapter that retrieves and adapts content from any JDBC-enabled data source.

stylesheet

An XSLT (eXtensible Stylesheet Language Transformations) instance that implements content presentation for XML documents. Portal-to-Go transformers can be either XSLT stylesheets or Java programs.

target format

The format required to deliver data to a specific type of client device.

Thin HTML

A minimal version of HTML implemented by a transformer in the starter Portal-to-Go repository. Thin HTML does not include support for frames, JavaScript, or other advanced features.

transformer

A Portal-to-Go object that converts content returned by Portal-to-Go adapters. Result transformers convert Adapter Result documents into Simple Result documents. Device transformers convert Simple Result documents into the target format.

TTML

Tagged Text Mark-up Language. A lightweight version of HTML suitable for most PDAs.

user agent

An object that associates an end user with a device type.

user group

A Portal-to-Go object that represents a set of users that are grouped together based on common criteria such as interests, subscription level, or geographic location.

VoxML

A markup language that enables the use of voice to interface with applications.

WAP

Wireless Application Protocol. A wireless standard from Motorola, Ericsson, and Nokia for providing cellular phones with access to email and text-based Web pages. WAP uses Wireless Markup Language (WML).

Web Integration adapter

An adapter that retrieves and adapts Web content using WIDL files to map the source content to Portal-to-Go XML.

WIDL

Web Interface Definition Language. A meta-data language that defines interfaces to Web-based data and services. WIDL enables automatic and structured Web access by compatible applications.

WIDL file

A file written in Web Interface Definition Language that associates input and output parameters with the source content that you want to make available in a Portal-to-Go service.

WML

Wireless Markup Language. A markup language optimized for the delivery of content to wireless devices.

XML

Extensible Markup Language. A flexible markup language that allows tags to be defined by the content developer. Tags for virtually any data item, such as product, sales representative, or amount due, can be created and used in specific applications, allowing Web pages to function like database records.

XSLT

Extensible Stylesheet Language Transformations. A stylesheet format for XML documents.

Index

A

ADAP element, 8-23
ADAP_LIST element, 8-23
Adapter Result format, 5-14, 8-4
AdapterDefinition class, 9-10
adapters, 1-4, 2-2

- arguments, 9-11
- configuration parameters, 9-14
- creating, 9-3
- deleting, 9-14
- filtering output, 9-11
- importing into the repository, 9-13
- modifying, 9-15
- parameters, 9-4
- sample class, 9-4

Address, user agent parameter, 6-5
administrator, user role, 6-1
adminUserLogin.jsp file, 7-7
AGEN element, 8-27
AGEN_LIST element, 8-27
agents. *See* user agents.
aliases, 2-3, 5-24

- creating, 4-8, 5-24
- parameters, 5-25

Anonymous, 6-1
arguments, adapter, 9-11
asynchronous notifications, 12-1
authenticate method, 6-13
authenticating users, 6-13

B

beginTag parameter, 5-15

BOMA element, 8-26
bookmarks, 2-4, 5-26

- creating, 5-26
- parameters, 5-27

C

cache implementation, 2-6
caption, parameter property, 5-11
chained services, 5-14, 8-4
chained services, creating, 4-15
checkUser method, 6-13
client device type

- recognizing, 11-5

configuration files, 2-4
connect string

- storing, 4-7
- system parameter, 2-5

cost, master service parameter, 5-4
createInit method, 9-10
createInput method, 9-11

D

database sessions, specifying minimum, 2-5
debug flag, setting, 12-5
designer, user role, 6-1
device recognition, 11-5

- class, 2-7

device transformer

- Java sample, 10-4
- specifying in a master service, 5-19
- XSLT sample, 10-7

Device, user agent parameter, 6-5

- devices, supporting new, 11-6
- Display Name, user parameter, 6-3
- DOM API support, 8-2
- Download utility, 8-33

E

- edit service window, Personalization Portal, 7-9
- edService.jsp file, 7-9
- endTag parameter, 5-15
- error
 - logging, 2-8
 - messages, 2-8
- error page parameter, 11-4
- EXT_ATTR element, 8-22
- extended attributes, 8-22
- External ID, user parameter, 6-3

F

- filtering adapter output, 9-11
- FOLD element, 8-24
- folders, 2-3
 - creating, 4-8
 - parameters, 5-23
- format method, 10-6
- FTP server, 12-7
- Ftp.properties, 12-7

G

- general parameters, master services, 5-5
- getAdapterDefinition method, 9-4, 9-10
- getInitDefinition method, 9-4, 9-10

H

- HDML, support for, 1-6
- HREF, bookmark parameter, 5-27
- HTTP error, when using a phone simulator, 4-13
- HTTP user agent parameter, 11-5

I

- implementation walkthroughs, 4-1
- init method, 9-8

- initialization parameters, adapter, 5-6
- input parameters, adapter, 5-10
- invoke method, 9-9

J

- JDBC driver, setting, 2-5
- JNDI naming, in service links, 8-4
- jobs
 - editing in Personalization Portal, 7-13
 - processing sequence, 12-1
- JTRA element, 8-22

L

- LDEV element, 8-22
- LDEV_LIST element, 8-22
- LINK element, 8-25
- linked services, 4-15
- LoadXML utility, 8-31
- log listeners, creating, 2-6, 13-2
- logging
 - error, 2-8
 - levels, 2-6
 - specifying the implementing class, 2-6
 - transaction, 2-7, 2-9
- logical devices, 2-3, 11-1
 - deleting, 11-5
 - login page, 11-3
 - modifying, 11-4
 - parameters, 11-2
- login page for a device, 11-3
- loginAuthenticate.jsp file, 7-7
- login.jsp file, 7-7
- LogListener class, 13-2

M

- MailSender class, 12-2
- MAST element, 8-26
- master services, 2-3
 - chained, 5-14, 8-4
 - configuring, 5-3
 - cost parameter, 5-4
 - creating, 4-7, 5-2 to 5-5

- deleting, 5-19
- general panel parameters, 5-5
- init parameters, 5-6
- input parameters, 5-10
- modifying, 5-5

MIME type parameter, 11-2

monitoring Portal-to-Go, 12-5

N

National Language Support (NLS), 7-17

needsURLCaching attribute, 8-22

notification, 12-1

notification engine, 12-2

notification listener, creating a, 13-2

NotificationListener interface, 12-2, 13-2

Notification.properties, 2-4, 12-4, 13-2

- parameters, 12-3

O

objects

- refreshing attributes, 12-5
- shutting down, 12-5

ordering services, 5-4

output parameters, 5-18

OutputArgument interface, 9-11

OutputType parameter, 5-13

P

PanamaObjects element, 8-17, 8-19

PapzMain.jsp file, 7-7, 7-8

parameters, adapter, 9-4

PAsection parameter, 5-12

Password Hint, user parameter, 6-3

Password, user parameter, 6-3

passwords

- algorithm for creating, 2-7
- changing in Personalization Portal, 7-15

Personalization Portal

- accessing, 4-12
- job edit sequence, 7-14
- login sequence, 7-7
- password edit sequence, 7-16

- service edit sequence, 7-11
- service editing, 7-9

PGRP element, 8-20

PGRP_LIST element, 8-19

phone simulator, using a, 4-12

PL/SQL procedures in services, 5-8

preference files, user, 7-15

preferences.xml, 2-8

probe monitor, 12-5

properties, system, 2-5 to 2-7

property files, 2-4

- rereading, 11-6

Provisioning format, 8-17

- content model, 8-17
- purpose, 8-3

provisioning systems, integrating, 6-12

ProvisioningHook interface, 6-13

Provisioning.properties, 2-4

PSRV_LIST element, 8-24

PUSR element, 8-17, 8-20

PUSR_LIST element, 8-17, 8-20

Q

query form, user, 6-3, 6-8

R

Raw Result format. *See* Adapter Result format

recognition, device, 11-5

repository connect string, 2-5, 4-7

Repository XML, 8-3, 8-19

result transformer

- sample, 10-8
- specifying in a master service, 5-18

RMI server, 12-6

Rmi.properties, 2-4

roles, user, 6-1

S

SAX API support, 8-2

sequence field, 5-4

Service Designer, 3-1

- starting, 3-2, 4-7

- walkthrough, 4-7
- ServiceRequest class, using the, 9-9
- Servlet adapter, 1-5
 - creating a service with, 5-19
- Short Message Service support. *See* SMS support.
- Simple Result format, 8-5
 - content model, 8-5
- SimpleCol element, 8-14
- SimpleContainer element, 8-7
- SimpleForm element, 8-10
- SimpleFormItem element, 8-11
- SimpleFormOption element, 8-12
- SimpleFormSelect element, 8-12
- SimpleMenu element, 8-9, 8-15
- SimpleMenuItem element, 8-9
- SimpleResult element, 8-6
- SimpleResultToText method, 10-6
- SimpleRow element, 8-14
- SimpleTable element, 8-13
- SimpleTableBody element, 8-14
- SimpleTableHeader element, 8-13
- SimpleText element, 8-8
- SimpleTextItem element, 8-8
- SMS support, 12-4
- SmsSender class, 12-2
- SQL adapter, 1-5
 - initialization parameters, 5-7
 - input parameters, 5-14
- SQL service, creating a, 4-13
- SQLTYPE parameter
 - sample, 4-14
- starting the Service Designer, 3-2, 4-7
- STATEMENT parameter
 - sample, 4-14
- stored procedures, calling in services, 5-8
- stripLevel parameter, 5-15
- stripper adapter, 1-5, 9-15
 - extending, 9-16
 - initialization parameters, 5-8
 - input parameters, 5-14
 - strip levels, 9-15
- Strip.properties, 2-4, 9-15
- System.properties, 2-5 to 2-7, 13-1

T

- tableView.jsp file, 7-8
- testing services, 4-12
- Tiny HTML, support for, 1-6
- TRAN_LIST element, 8-21
- transaction logging, 2-7, 2-9
- transform method, 10-7
- transformers, 1-6, 2-3, 10-1
 - configuration parameters, 10-10
 - creating, 10-3
 - deleting, 10-11
 - importing into the repository, 10-10
 - modifying, 10-11
 - sample Java, 10-4
 - XSLT sample, 10-7
- treeView.jsp file, 7-8

U

- Upload utility, 8-33
- URL adapter, 1-5
- URL adapter parameter, 5-15
- URL caching, specifying for devices, 11-3
- user agents, 2-3, 7-16, 11-5
 - creating, 6-3
 - parameters, 6-5
- User Customizable, parameter property, 5-12
- user groups, 2-3, 6-8
 - adding users, 6-10
 - creating, 4-10, 6-8
 - deleting, 6-10
 - parameters, 6-8
 - removing users, 6-11
 - service access, 6-10
- user preference files, 7-15
- User Root parameter, 6-3
- useragent.properties, 2-4, 11-5
- UserAuthenticationHook interface, 2-7, 6-13
- users, 2-3, 6-2
 - creating, 4-10, 6-2
 - deleting, 6-6
 - finding in the repository, 6-3
 - integrating provisioning systems, 6-12
 - modifying, 6-7

parameters, 6-3
XML definitions, 8-17

V

VoxML, support for, 1-6

W

walkthroughs, 4-1
Web Integration adapter, 1-5
 initialization parameters, 5-6
 input parameters, 5-13
Web Integration Server, 4-6
Web Interface Definition Language files. *See* WIDL services.
WIDL Interface parameter, 5-7
WIDL services
 creating, 4-2
 publishing, 4-6
WML, support for, 1-6

X

XML Editor, 8-28
XML parser, 8-2
XTRA element, 8-21

