Oracle9*i* Application Server Wireless Edition

Developer's Guide

Release 1.1

January 2001

Part No. A86700-01

ORACLE

Oracle9*i* Application Server Wireless Edition Developer's Guide, Release 1.1

Part No. A86700-01

Contents

7 Using the Data Model API

8 Using the Location API

List of Figures

List of Tables

Send Us Your Comments

Oracle9*i* Application Server Wireless Edition Developer's Guide, Release 1.1

Part No. A86700-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. 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?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: iasdocs_us@oracle.com
- Postal service:
  Oracle Corporation
  Oracle Mobile and Wireless Products
  500 Oracle Parkway, Mailstop 4OP6
  Redwood Shores, California 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

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

Preface

This guide discusses how you can use the Wireless Edition to develop and deliver mobile services. It includes these chapters:

Chapter 1, "Introduction"	Provides an overview of Oracle9*i* Application Server Wireless Edition.
Chapter 2, "Creating Adapters"	Describes how to create Oracle9*i* Application Server Wireless Edition adapters.
Chapter 3, "Creating Transformers"	Describes how to create Oracle9*i* Application Server Wireless Edition transformers.
Chapter 4, "Rebranding the Personalization Portal"	Describes the Web interface that users access to customize their mobile portals.
Chapter 5, "Using the Personalization Portal API"	Describes The Personalization Portal API classes and how they provide a streamlined way for portal customization.
Chapter 6, "Using the Runtime API"	Explains how the Wireless Edition Runtime processes events and how developers can use the hooks in the Runtime API.
Chapter 7, "Using the Data Model API"	Describes the class hierarchy and the semantics of each class in the Data Model API. Explains how adapters can use the API.

Chapter 8, "Using the Location API"	Describes how by using the interfaces in the Location API developers can integrate location services (from geocoding, mapping, routing, and yellow pages third-party providers) into Wireless Edition applications without having to write custom interfaces for each service.
Chapter 9, "Working with Wireless Edition XML"	Describes the XML formats that Oracle9*i* Application Server Wireless Edition uses to represent service content and its internal objects.
Appendix A, "Simple Result DTD Reference"	Describes the content of the Simple Result DTD; the elements, the usage, the attributes, the children, and the DTD declarations.
Appendix B, "Runtime System Variables"	Describes the system variables that are recognized by the runtime application.

1 Introduction

This guide discusses how you can use the Oracle9*i* Application Server Wireless Edition to develop and deliver mobile services. It explains how to create adapters and transformers, customize your mobile portals at various levels (JavaServer Pages, Portal API, Data Model API, and Runtime API), extend and customize the functional components in the Wireless Edition, and work with the XML formats that the Wireless Edition uses.

Each section of this document presents a different topic. These sections include:

1.1 Creating Adapters

The Wireless Edition architecture is designed to adapt content from Web sites, Internet applications, and databases, into an XML representation. The XML representation is transformed through the use of XSL stylesheets to a markup language specific to a particular device.

The Wireless Edition provides adapters that retrieve any content. You can also create your own adapters by implementing the `RuntimeAdapter` interface. The newly created adapter can be a self-contained application, or it can invoke external

applications. For details on how to create new adapters, see Chapter 2, "Creating Adapters". For information on how an adapter can use the interfaces in the Wireless Edition Runtime API and Data Model API, see Chapter 6, "Using the Runtime API" and Chapter 7, "Using the Data Model API", respectively.

1.2 Creating Transformers

Implementing the ServiceContext interface, which contains the input and output parameters of a service in addition to the AdapterResult and SimpleResult, provides the XML input for the transformers. The device transformers convert SimpleResult into the format of the target device.

The implementation of the Transformer interface is public, enabling any level of customization of this process, both for custom and existing adapters. For any custom adapter, it is possible to create a new ResultTransformer, provided that the output from the transformer is Wireless Edition DTD-compliant XML. This XML is then converted by a device transformer to a markup language format that is appropriate for the mobile device.

The Wireless Edition provides generic device transformers for several target formats, such as CHTML, HDML, HTML, MML, VoiceXML, VoxML, and WAP (WML). It is possible to customize specific device transformers or create new ones. This means that device-specific transformations can be implemented. To support a new markup language or a new device type, only the device transformer needs to be modified, but the application itself is unaffected.

As described in Chapter 7, "Using the Data Model API", the implementation of the Device interface enables the definition of the target logical device protocol (for example, WML11, SMS, or EMAIL), as well as the specification of the physical characteristics of the target logical device that can be used by the adapters and the transformers (for example, screen width and height, screen columns and rows, and number of softkeys). There are Runtime system variables that can access this information, as described in Chapter 6, "Using the Runtime API".

For more information on creating transformers, see Chapter 3, "Creating Transformers". For more information on working with Wireless Edition XML, see Chapter 9, "Working with Wireless Edition XML".

1.3 Rebranding the Wireless Edition Personalization Portal

The Wireless Edition 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, XSL stylesheets, 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.

For more information on rebranding the Personalization Portal, see Chapter 4, "Rebranding the Personalization Portal".

1.4 The Runtime API

The Runtime API consists of three public Java packages that provide interfaces for the following:

- Examination of the status of the core objects in the Runtime execution (`oracle.panama.rt`).

  The essential Runtime core objects are:

  - Request: the service invocation specification
  - Response: the result of a request execution
  - Session: the duration and context of a connection
  - ServiceContext: the context in which the request is being executed
  - ManagedContext: the private context of an adapter

- Monitoring the Runtime execution sequence, based on the event model (`oracle.panama.rt.event`)
  - RequestListener observes the enabled Request Event.
  - ResponseListener observes the enabled Response Event.
  - SessionListener observes the enabled Session Event.

- Extending the Runtime essential replaceable components (`oracle.panama.rt.hook`) by using hooks and policies
  - Hook is the main extension mechanism.
  - The default implementations of hooks are published as policies.
  - Policies can be reused by the customized hook.
  - The new customized hooks, which implement the respective interfaces and the singleton pattern, are registered in the appropriate entries in the **System.properties** file.

1.4.1 Hooks

One set of the interfaces in the Runtime API, which is contained in the package `oracle.panama.rt.hook`, specifies the hooks that can be used by application developers for their customized plug-in modules. For example, The ListenerRegistrationHook registers listeners. Application developers can implement this hook interface for a customized listener registration module that lets the listeners selectively observe the event sources. A reusable custom listener registration policy may subscribe the listeners only to the requests for the Web Integration Adapter services. Such a listener may add business rules to the Runtime controller.

For more information on how to use hooks, see Chapter 6, "Using the Runtime API".

1.5 The Data Model API

You can create, delete, modify, and query Wireless Edition persistent objects by implementing the public Java interfaces in the Data Model API. Developers of services, adapters, and transformers can implement these interfaces to develop stand-alone applications which manipulate a Wireless Edition persistent object.

For example, a User can belong to multiple groups. The actual user can access the services that are accessible to the groups to which the user belongs. However, the implementation of the User interface can access another provisioning system to manage the information about the current actual user and specify only particular services that are available to that user.

Any service can be assigned as accessible to a group. User attributes can use LocationMark and AutoLocate for location-based services.

The Device interface in the Data Model API provides for defining the target logical device protocol (for example, WML11, SMS, or EMAIL), as well as specifying the physical characteristics of the target logical device (logical device attributes) that can be used by the adapters and the transformers (for example, screen width and height, screen columns and rows, and number of softkeys). There are Runtime system variables that can access this information.

For more information about the Data Model API, see Chapter 7, "Using the Data Model API".

1.6 The Location API

The Wireless Edition provides a set of location application component APIs that enable developers to include geocoding, location marks, routing, and business directory (yellow pages) components into their Wireless Edition applications. By using these APIs, developers can integrate location services from geocoding, mapping, routing, and yellow page third-party providers into Wireless Edition applications without having to write custom interfaces for each service.

Developers can prioritize services based on quality, availability, or cost. The location application component APIs also include a wrapper function that maps location services providers to the APIs.

For more information on how to use the location API, see Chapter 8, "Using the Location API".

1.7 Wireless Edition XML

XML is a markup language for documents containing structured information. In such documents, the role that an element of information plays in that document is significant. Therefore, it is possible in documents that have some structure to separate content (for example, words and pictures) from indications of what role that content plays (for example, the content in a section heading carries a different meaning from the same content in a database table or a footnote). XML is an effective mechanism for identifying structures in a document, which makes it useful for exchanging data between diverse platforms.

In Chapter 9, "Working with Wireless Edition XML", there are descriptions of the Oracle XML parser, the Wireless Edition XML formats — the SimpleResult format and the AdapterResult format, the XML structure of the Wireless Edition repository, and XML tools, such as LoadXml and upload and download utilities.

Appendix A, "Simple Result DTD Reference" provides a detailed reference for the SimpleResult DTD.

2

Creating Adapters

This document describes how to create and manage Java classes that implement Wireless Edition adapters. Each section of this document presents a different topic. These sections include:

2.1 What Is a Wireless Edition Adapter?

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

2.1.1 Using Wireless Edition Adapters

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

The Wireless Edition provides adapters that retrieve any content, including Web, database, and spatial content. The *Oracle9i Application Sever Wireless Edition Implementation Guide* describes the adapters provided by the Wireless Edition, and explains how to create services that use them.

2.1.2 Implementing Wireless Edition Adapters

You create your own Java adapters for the Wireless Edition by implementing the `RuntimeAdapter` 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 the Wireless Edition. The adapter that you create must simply expose the entry points and return the value types that the Wireless Edition expects.

2.1.3 Adapter Content Formats

Adapters return content that conforms to a Wireless Edition content DTD. The returned content can be in one of the following formats:

- SimpleResult
- AdapterResult

In most cases, adapters return content that conforms to the SimpleResult DTD. The SimpleResult DTD contains elements that represent the components of an abstract user interface, such as tables, text, forms, and menus.

Adapters can also return content in the AdapterResult format. The AdapterResult format is an intermediary, user interface-independent content format. You can use it to pass raw data between Wireless Edition components, such as chained services.

Since device transformers operate only on the SimpleResult format, a ResultTransformer associated with a master service must convert an AdapterResult document to the SimpleResult format before the content can be delivered.

Note: The target attributes in the SimpleResult XML that is returned by the adapter should not be URL encoded. However, the target attributes in the SimpleResult XML document, which are fetched by the URL Adapter (either the built-in adapter or a modified version of it) should be URL encoded.

2.2 Creating Adapters

Generally, an adapter performs the following functions:

1. Connects to a data source.

2. Retrieves content.
3. Converts the content to a Wireless Edition XML format.

The adapter should know the content source format and how to map it to a Wireless Edition results format.

Follow these rules to create an adapter:

1. Implement the `oracle.panama.rt.RuntimeAdapter` interface.
2. Provide an empty, default constructor.
3. Make the class thread-safe.
4. Implement the following methods in the `RuntimeAdapter` interface:
   - `init()`
   - `invoke()`
   - `getInitArguments()`
   - `getInputArguments()`
   - `getOutputArguments()`
   - `destroy()`

The master service calls the `init()` function once, the first time the adapter is invoked. The `init()` function performs initialization functions for the adapter. The master service calls the `invoke()` function every time the adapter is invoked.

The `getInitArguments()`, `getInputArguments()`, and `getOutputArguments()` methods enable the Wireless Edition to create an adapter definition for an adapter. The `getInitArguments()` method gets the initialization arguments for the MasterService and returns the arguments or null if no arguments exist. The `getInputArguments()` method gets the input arguments for the MaterService. The `getOutputArguments()`method gets the output arguments for the MasterService. The `destroy()` method destroys the provider, thereby releasing the resources that were implemented by the adapter.

The Wireless Edition calls the methods in the following order:

1. Default constructor
2. `getInitArguments()`
3. `init()`
4. `getInputArguments()`

5. `getOutputArguments()`
6. `invoke()`
7. `destroy()`

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

2.2.1 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.

The complete source code for the sample appears in Section 2.4, "HelloAdapter Source Code".

The reference numbers in brackets that appear in the discussion refer to the corresponding section in the complete listing of the source code.

Sample: HelloAdapter

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

When invoked, the adapter returns these XML elements which conform to the SimpleResult DTD:

```
<SimpleResult>                [4]
  <SimpleContainer>
   <SimpleText title="Oracle Portal-to-Go Server Hello Sample">
      <SimpleTextItem name="message" title="Portal-to-Go says:">
         <greeting> <name>!
      </SimpleTextItem>
   </SimpleText>
  </SimpleContainer>
</SimpleResult>
```

The following sections describe each method in the sample adapter.

2.2.1.1 init()

The `init()` method initializes the adapter. Every adapter in the Wireless Edition must implement an `init()` method. The Wireless Edition 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 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. The RuntimeAdapterHelper class contains methods for creating input arguments and output arguments. Following initialization, the `createArguments()` and the `createOutputArguments()` methods are called out. They return the input arguments and the output arguments as follows:

```
Public void init (Arguments args) throws AdapterException {  [3]
    synchronized (this) {
        initialized = true;
       greeting = args.getInputValue ( GREETING );
    }
}
```

2.2.1.2 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 `ServiceContext`. It returns an XML element that holds the result. See the source code section with the reference number **[4]** in the complete source code listing for HelloAdapter in Section 2.4, "HelloAdapter Source Code".

Upon receiving an end user request, the Wireless Edition Request Manager creates a `ServiceContext` object. The `ServiceContext` contains any parameter values specified by the user, service alias, or master service. The `ServiceContext` 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(doc, "SimpleResult");
```

Then, it creates the container element:

```
Element container = XML.makeElement(result, "SimpleContainer");
        result.appendChild (container);
```

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

```
Element st = XML.makeElement(container, "SimpleText");
st.setAttribute ("title", "Oracle Portal-to-Go Server HelloAdapter Sample");
container.appendChild (st);
```

As defined in the SimpleResult 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(st, "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.getInputArguments().getInputValue( NAME );
  Text txt = XML.makeText(sti, greeting + " " + name + "!");
  sti.appendChild (txt);
  return result;
```

2.2.1.3 getInitArguments()

Every adapter must implement `getInitArguments()`. This method enables a master service to determine the initialization parameters that are used by the adapter.

The `RuntimeAdapter` class encapsulates the adapter parameters. In the `getInitArguments()` method, you get the initialization parameters in the `Adapter` object and return the initialization arguments.

The numbers in bold that appear on the right side of the source code excerpts below refer to where the excerpt is located within the complete source code listing in Section 2.4, "HelloAdapter Source Code".

```
private Arguments initArgs = null;            [5]
public Arguments getInitArgument() throws AdapterException {
  if (initArgs == null) {
      synchronized (this) {
        if (initArgs == null) {
```

```
            initArgs = RuntimeAdapterHelper.createArguments();
            // Any init parameters would be set here..
           Argument initArg = initArgs.createInput(GREETING, "Greeting phrase",
null, true );
                    initArg.setType(ArgumentType.SINGLE_LINE);

       }
     }
  }
  return initArgs;
}
```

2.2.1.4 getInputArguments()

```
private Arguments inputArgs = null;          [6]
public Arguments getInputArguments()  throws AdapterException{
  if (inputArgs == null ) {
      synchronized (this) {
        if (inputArgs == null) {
            inputArgs = RuntimeAdapterHelper.createArguments();
            Argument inputArg = inputArgs.createInput(NAME, "Name to greet",
null ,true);
                    inputArg.setType(ArgumentType.SINGLE_LINE);
        }
      }
  }
  return inputArgs;
}
```

2.2.1.5 getOutputArguments()

```
private OutputArguments outputArgs = null;             [7]
public OutputArguments getOutputArguments()  throws AdapterException{
        if (outputArgs == null) {
           synchronized (this) {
               outputArgs = RuntimeAdapterHelper.createOutputArguments();
           }
        }
        return outputArgs;
   }
```

```
    public void destroy() { }

}
```

2.2.2 Managing Arguments

The Wireless Edition 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.

2.2.2.1 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.

2.2.2.2 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();
...
```

2.3 Importing an Adapter into the Repository

After you create and compile the class that implements the adapter, you import the adapter into the Wireless Edition repository.

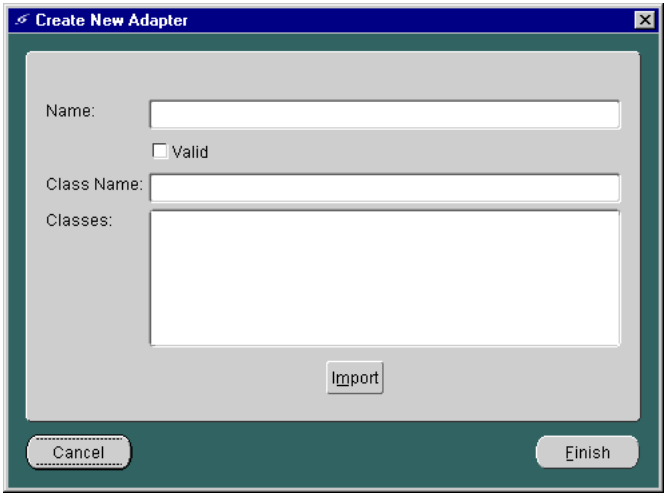
Note: You do not have to import an adapter into the repository if the adapter is in the CLASSPATH.

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 Oracle9*i* Application Server Wireless Edition 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. Start the Service Designer.
2. Right-click Adapters in the Service Designer Repository tree view.
3. Click Create New Adapter. The Create New Adapter dialog box appears.

Figure 2–1 Create New Adapter



4. Fill in the fields for each parameter. Select the box next to Valid to specify that the adapter is available to services. If the box is not selected (that is, if the

adapter is invalid), then the adapter is unavailable, and all master services that use the adapter are invalid.

The form includes the following parameters:

Table 2–1 Adapter 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.

2.3.1 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 Service Designer repository tree view.
2. Right-click the adapter and click Delete.

2.3.2 Modifying Adapters

To modify an adapter:

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

2.4 HelloAdapter Source Code

The complete source code for HelloAdapter is listed below. Each section of the code is numbered on the right side of the text (for example, [1]) to assist in mapping the description in Section 2.2.1, "Sample Adapter Class" to the source code in its entirety.

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

package sampleadapter;
import org.w3c.dom.Element;                                    [1]
import org.w3c.dom.Document;
import org.w3c.dom.Text;

import oracle.panama.Argument;
import oracle.panama.Arguments;
import oracle.panama.OutputArguments;
import oracle.panama.ArgumentType;
import oracle.panama.rt.ServiceContext;
import oracle.panama.adapter.RuntimeAdapter;
import oracle.panama.adapter.AdapterException;
import oracle.panama.adapter.RuntimeAdapterHelper;
import oracle.panama.core.util.XMLUtil;
import oracle.panama.core.xml.XML;

public class HelloAdapter implements RuntimeAdapter {        [2]

    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 { [3]
    synchronized (this) {
        initialized = true;
      greeting = args.getInputValue( GREETING );
    }
}

/**                                                              [4]
 *  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 (ServiceContext sr) throws AdapterException {

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

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

    // Create SimpleContainer element
    Element container = XML.makeElement(result, "SimpleContainer");
    result.appendChild(container);

    // Create SimpleText element
    Element st = XML.makeElement(container, "SimpleText");
    st.setAttribute ("title", "Oracle Portal-to-Go Server HelloAdapter
                                                  Sample");
    container.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.getInputArguments().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 Arguments initArgs = null;     [5]
public Arguments getInitArguments() throws AdapterException {
  if (initArgs == null) {
      synchronized (this) {
        if (initArgs == null) {
            initArgs = RuntimeAdapterHelper.createArguments();
            // Any init parameters would be set here..
           Argument initArg = initArgs.createInput(GREETING, "Greeting phrase",
                                                   null, true );
                       initArg.setType(ArgumentType.SINGLE_LINE);

             }
           }
       }
       return initArgs;
    }

    private Arguments inputArgs = null;       [6]
    public Arguments getInputArguments()  throws AdapterException{
      if (inputArgs == null ) {
          synchronized (this) {
            if (inputArgs == null) {
                inputArgs = RuntimeAdapterHelper.createArguments();
               Argument inputArg = inputArgs.createInput(NAME, "Name to greet",
                                                         null ,true);
                      inputArg.setType(ArgumentType.SINGLE_LINE);
            }
          }
```

```
        }
        return inputArgs;
    }

    private OutputArguments outputArgs = null;  [7]
    public OutputArguments getOutputArguments()  throws AdapterException{
        if (outputArgs == null) {
           synchronized (this) {
               outputArgs = RuntimeAdapterHelper.createOutputArguments();
           }
        }
        return outputArgs;
    }

    public void destroy() { }

}
```

3

Creating Transformers

This document describes how to create and manage Wireless Edition transformers. Each section of this document presents a different topic. These sections include:

3.1 Overview

Wireless Edition transformers are Java programs or XSLT stylesheets that convert a document into either the target format or another Wireless Edition format.

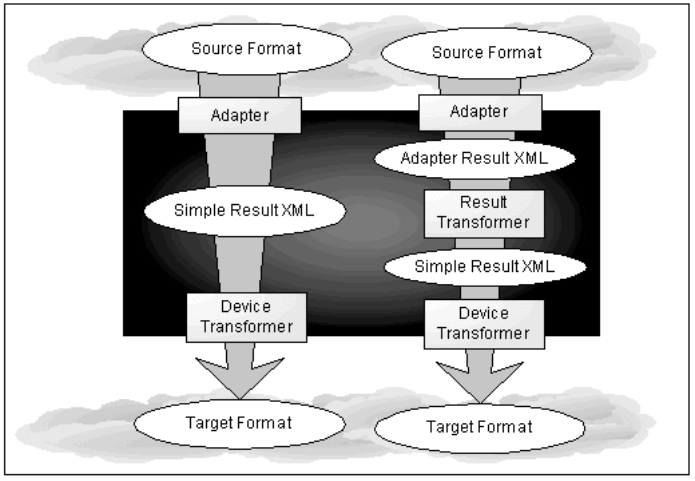
The Wireless Edition supports two types of transformers:

- 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 Wireless Edition 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 the Wireless Edition can deliver it to the user.

Figure 3–1 Wireless Edition Transformers



Device transformers convert Simple Result documents into the format of the target device. You can use two types of device transformers in the Wireless Edition:

- Default transformer
- Custom transformer

The Wireless Edition requires you to associate a transformer to each logical device.

A logical device is a repository object that represents either a physical device, such as an Ericsson mobile phone, or an abstract device, such as an email server. Logical devices represent the interface between transformers and the target devices or applications. For more information on logical devices, see *Oracle9i Application Server Wireless Edition Implementation Guide.*

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.

The Wireless Edition 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 transformer 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 and can only be used for that device.

3.2 Wireless Edition Transformers

The Wireless Edition publishes device transformation rule files so that anyone can create support for any type of device and markup language.

The Wireless Edition initial repository includes transformers for several target formats, such as CHTML, HDML, HTML, MML, VoiceXML, VoxML, and WAP (WML).

By modifying the transformers provided with the Wireless Edition, or by 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 that is 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 Wireless Edition repository, maps several Simple Result elements as follows:

Table 3–1 Tiny HTML Transformer Mapping

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

Similarly, when you create a new 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 the Wireless Edition transformers as either Java transformers or XSLT stylesheets.

3.3 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.

3.4 Java Transformers

Java transformers implement the `RtTransformer` interface. The transformers convert the XML structure in the SimpleResult document into a device-specific markup language. The classes that implement the `RuntimeAdapter` interface can use the `ServiceContext` interface in the `oracle.panama.rt` package. The `ServiceContext` interface contains the input and output parameters for the service. It also contains the AdapterResult and SimpleResult, which are externalized as an XML document.

The `ServiceContext` XML document is the input document for the transformer. The XSL stylesheets for the transformers must be written against the DTD for the `ServiceContext` XML document. The implementation of the `RtTransformer` interface must provide a default constructor (that is, a constructor without arguments). Also, the implementation should return a `String` object, which is the result document that contains a device-specific markup language, such as WML.

3.5 Creating a Transformer

You can create a Java transformer or an XSL transformer. When you create a transformer, make sure that you have accomplished the following first:

1. Created a master service that is associated with the transformer.
2. Created an adapter definition for that service. The AdapterDefinition should return a document with a root named <AdapterResult>.
3. Specified a PAsection with a value equal to the name of the ResultTransformer that you want to apply.

4. Selected a logical device, or created a new logical device in the repository, that is associated with a default device transformer or a custom device transformer.
5. Created a unique name for the transformer. The name is stored in the transformer folder of the repository tree.

A result transformer must render the AdapterResult document in SimpleResult XML format, which is based on the SimpleResult DTD. The SimpleResult XML provides the input to the device transformer, which then converts the service in terms of the particular type of device for which the service is targeted.

If the format of the target device requires a device transformer that is different than the available default transformers, then you should create a custom device transformer.

Use the Create New Transformer form in the Service Designer to create a transformer in the repository. For information on how to use the form, see Section 3.6.1, "Creating a Transformer in the Repository".

3.5.1 Creating an XSL Transformer

To create an XSL transformer, start your code with the following line after the header:

```
<xsl:template match="/">
```

In the processing sequence of the XSLT stylesheet, 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 `<xsl:apply-templates/>` 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 following transformer converts SimpleResult documents to plain text. It is the XSLT version of the Java transformer described in Section 3.5.3, "Creating a Java Transformer". This XSLT device transformer is included in the Wireless Edition initial repository.

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
      <xsl:apply-templates/>
   </xsl:template>
```

```
   <xsl:template match="SimpleTextItem | SimpleFormItem | SimpleMenuItem">
      <xsl:text/>
      <xsl:value-of select="."/>
   </xsl:template>
   <xsl:template match="SimpleRow">
      <xsl:text/>
      <xsl:for-each select="./SimpleCol">
         <xsl:text/>
         <xsl:value-of select="."/>
      </xsl:for-each>
   </xsl:template>
</xsl:stylesheet>
```

The sample code illustrates the processing sequence for XSLT stylesheets. After matching, or selecting, a SimpleResult element and processing the contents of that found 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.

This example 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 Wireless Edition element:

```
<SimpleTextItem>Scott</SimpleTextItem>
```

The XSLT segment would produce the following:

```
<P>
   Scott
</P>
```

3.5.2 Sample XSLT ResultTransformer

The following ResultTransformer segment shows how a ResultTransformer can enable chained services. It uses the AdapterResult of a user's selection to link to a target, which is the subsequent service in the chain.

```
<xsl:template match="AdapterResult">
   <SimpleResult>
     <SimpleContainer>
      <SimpleMenu name="Query Results">
         <xsl:for-each select="Table/Row">
            <xsl:if test="position() &#60; 8">
               <SimpleMenuItem>
                  <xsl:attribute name="target">
      ___REQUEST_NAME__?PAoid=__PAoid__
      &#38;PAsection=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>
    <SimpleContainer>
   </SimpleResult>
</xsl:template>
```

The XML parser processes the AdapterResult XML document using the ResultTransformer.

In the above example, the ResultTransformer includes the variables _REQUEST_NAME__ and _PAoid__. For more information about these and other Runtime variables, see Chapter 6, "Using the Runtime API".

- The variable _REQUEST_NAME represents the name of the JSP file that executes the request.
- The variable _PAoid represents the internal unique object ID for the service. The Wireless Edition Runtime will replace this parameter with the value automatically.

For example, given the following string:

http://www.*host*.com/ptg/rm?PAoid=250

- _REQUEST_NAME represents ptg/rm

- _SERVICE_NAME represents 250 (which could be a folder, an alias, a master service, or a link)

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"/>
```

3.5.3 Creating a Java Transformer

A Java transformer specifies a Java class transformer implementation. To create a Java transformer for the Wireless Edition, you implement the RtTransformer interface.

```
 /*$Copyright:
*             Copyright (c) 2000 Oracle Corporation all rights reserved
* $
*/
package oracle.panama.rt.xform;

import java.io.Writer;
import org.w3c.dom.Element;
import oracle.panama.PanamaException;

/**
 * Transform from a XML structure to a device specific content.
  */
public interface RtTransformer {

    /**
     * Transform the simple result XML document into a device specific markup
language.
     * @param element the <code>SimpleResult</code> XML Element to process.
     * @param out the output writer for the result
     */
    public void transform(Element element, Writer out) throws PanamaException;

}
```

The implementation must be thread-safe and it must provide a default constructor (that is, a constructor without arguments). The transformer should return a String object, which is the result document that contains a device-specific markup language.

The following section describes the components of an `RtTransformer` interface implementation.

3.5.4 Components of a Java Transformer

This section presents the components of a Java transformer. The transformer that is used in this section as an example converts SimpleResult XML 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 SimpleResult XML elements into another format, such as the particular format of the target device.

```
/*
 *
 * $Copyright:
 *             Copyright (c) 1999 Oracle Corporation all rights reserved
 * $
 */
package oracle.panama.core.xform;

import org.w3c.dom.NodeList;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import java.io.Writer;
import java.io.IOException;
import oracle.panama.PAPrimitive;
import oracle.panama.PanamaException;
import oracle.panama.model.Service;
import oracle.panama.rt.RequestFactory;
import oracle.panama.rt.Request;
import oracle.panama.rt.xform.RtTransformer;

/**
 * Transform from panama simple result to plain text. This transformation is
 * very simple and is intended to be used when sending SMS messages to cell
 * phones.
 *
 * @author pelarsso
 * @version $Revision: 1.9 $
 * @since PANAMA_10
 */
public class SimpleResultToText implements RtTransformer {

    /**
```

```
     * Empty constructor.
     */
    public SimpleResultToText() {}

    /**
     * @return null if the name is of zero length or null otherwise return the
name.
     */
    private String getName(String name) {
        return name == null || name.length() == 0 ? null : name;
    }

    /**
     * @return the name of this element, thename is wither the attribute value
of title or name in this order.
     */
    private String getName(Element el) {
        String attr;
        attr = getName(el.getAttribute("title"));
        if (attr == null) {
            attr = getName(el.getAttribute("name"));
        }
        return attr;
    }

    /**
     * @return the text node value of this element or an empty string if none
exists.
     */
    private String getTextValue(Element el) {
        Node n = el.getFirstChild();
        return (n != null && n.getNodeType() == Node.TEXT_NODE) ?
n.getNodeValue() : "";
    }
    /**
     * Format the element to a string.
     * @param el the XML element.
     * @return the string representation.
     */
    private String format(Element el) {
        if (el == null) {
            return "";
        }
        StringBuffer buf = new StringBuffer();
        String attr = getName(el);
```

```
        if (attr != null) {
            buf.append(attr);
            buf.append(": ");
        }

        buf.append(getTextValue(el));

        return buf.toString();
    }

    /**
     * Get the child where matching the name.
     *
     * @param element the parent element.
     * @param name    the name of the sub-element to search for.
     * @return        the first matching child or null if no match.
     */
    private Element getChildByName(Element element, String name) {
        Element rEl = null;
        for (Node node = element.getFirstChild(); node != null; node =
node.getNextSibling()) {
            if (node.getNodeType() == Node.ELEMENT_NODE &&
node.getNodeName().equals(name)) {
                rEl = (Element)node;
                break;
            }
        }
        return rEl;
    }

    /**
     * Transform the simple result XML document into a device specific markup
language.
     * @param element the <code>SimpleResult</code> XML Element to process.
     * @param out the output writer for the result
     */
    public void transform(Element element, Writer out) throws PanamaException {
        // Put in the request name as header
        Request req = RequestFactory.lookupRequest();
        Service service = req == null ? null : (Service)
req.getServiceContext().getService();
        StringBuffer buf = new StringBuffer((service == null) ? "" :
service.getName());

        // Get all elements in the first container.
```

```
        Element el = getChildByName(element, "Result");
        if (el != null) {
            el = getChildByName(el, PAPrimitive.TAG_SIMPLERESULT);
            if (el != null) {
                el = getChildByName(el, PAPrimitive.TAG_SIMPLECONTAINER);
                if (el != null) {
                    element = el;
                }
            }
        }

        NodeList list = element.getElementsByTagName("*");

        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));
            }
        }
        String result = buf.toString();
        try {
```

```
            out.write(result);
            out.flush();
        } catch (IOException ex) {
            throw new PanamaException(ex);
        }
    }
}
```

The following sections describe each method in the transformer.

3.5.4.1 SimpleResultToText()

The `SimpleResultToText()` method is the default constructor for instantiating the transformer. Any transformer you create must also provide an empty default constructor.

3.5.4.2 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
```

3.5.4.3 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.

3.6 Managing Transformers with the Service Designer

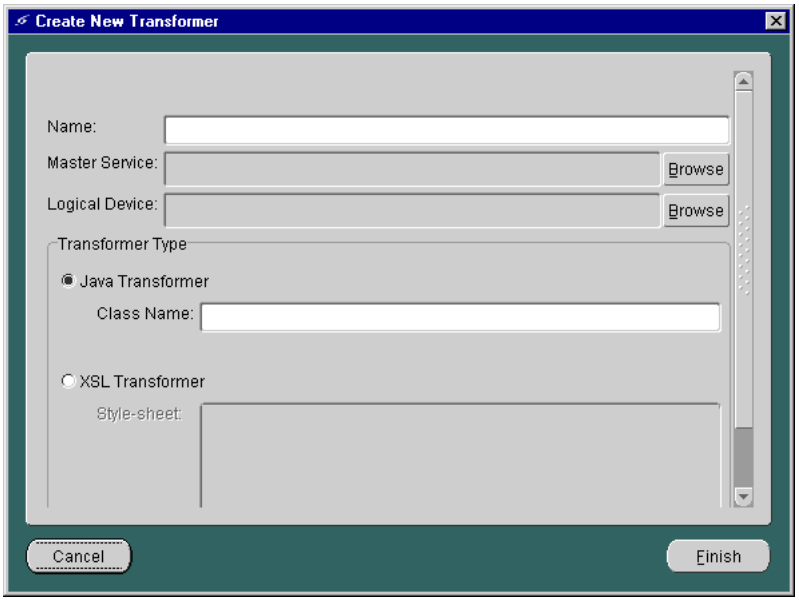
The Service Designer enables you to manage transformers in the Wireless Edition repository. Using the Service Designer, you can create and update transformers, and remove them from the repository.

3.6.1 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 Wireless Edition repository tree view.
2. Click Create New Transformer to display the first form in the sequence. The Create New Transformer dialog box appears.

Figure 3–2 Create New Transformer Form



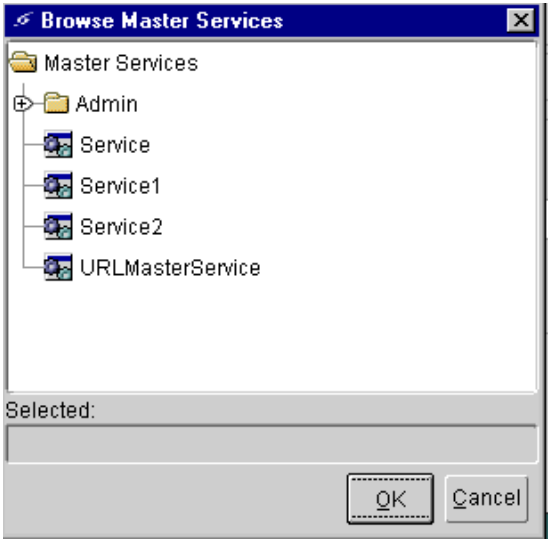
The form includes the following parameters:

Table 3–2 Transformer Parameters

Parameter	Value
Name	The transformer name. This must be a unique name in the transformer folder of the repository tree.
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.

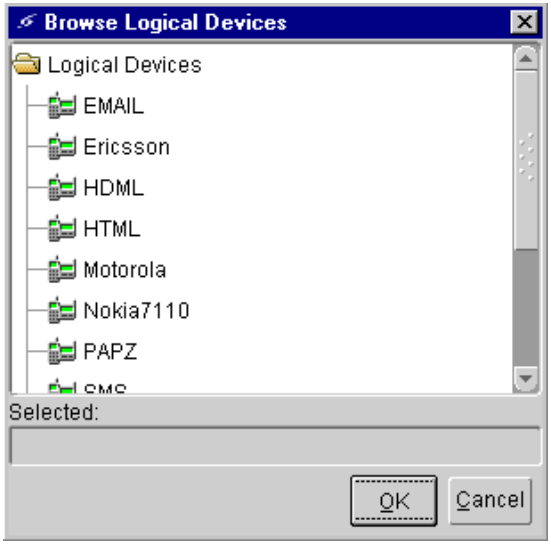
3. Enter the name of the transformer in the Name field.
4. Click the Browse button next to the Master Service field. The Browse Master Services dialog box appears. Select a service and click OK.

Figure 3–3 Browse Master Services



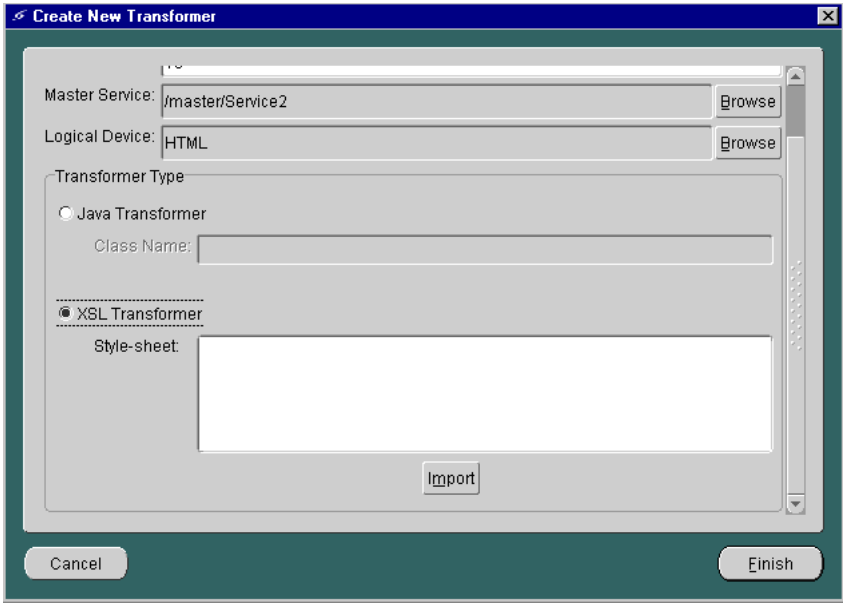
5. Click the Browse button next to the Logical Devices field. The Browse Logical Devices dialog box appears. Select a logical device and click OK.

Figure 3–4 Browse Logical Devices



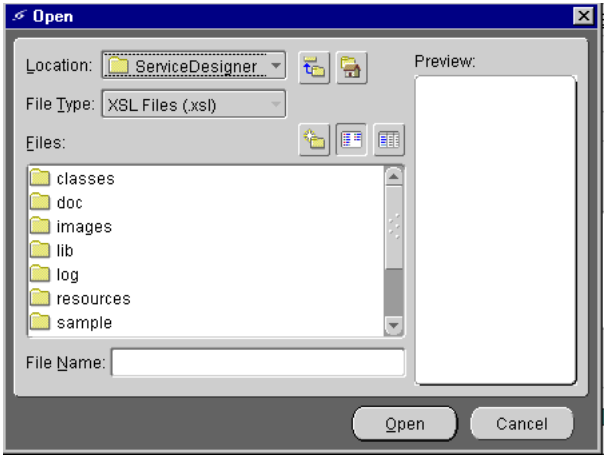
6. Select the transformer type that you want to use.
7. If you select a Java transformer, enter the class name of the transformer, then click Finish.
8. If you select an XSL transformer, you can do ***one*** of the following:
   a. Type the code for the XSL stylesheet in the field next to the Style-sheet parameter, then click Finish.
   b. Using a text editor, open an existing XSL stylesheet, copy the lines that you want to use, and paste them in the field next to the Style-sheet parameter in the Create New Transformer dialog box. Click Finish.
   c. Using the scroll bar in the Create New Transformer dialog box, scroll down to the bottom of the dialog box. Click the Import button to import an existing XSL stylesheet.

Figure 3–5 The Import Button in Create New Transformer Dialog Box



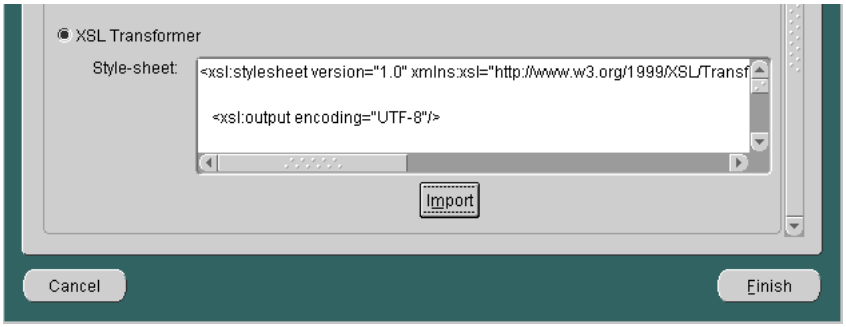
The Open dialog box appears, as shown in the following figure.

Figure 3–6 The Open Dialog Box



Navigate to find the XSL file that you want to use, then click Open. The contents of the imported XSL stylesheet appear within the Style-sheet field in the Create New Transformer dialog box. Click Finish. You have created a new transformer.

Figure 3–7 Imported XSL Stylesheet



3.6.2 Modifying a Transformer

To modify a transformer in the Wireless Edition repository, follow these steps:

1. Expand the Transformers folder in the Wireless Edition 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 Section 3.2, "Wireless Edition Transformers".

3.6.3 Removing a Transformer from the Repository

To remove a transformer from the repository:

1. Expand the Transformers folder in the Wireless Edition 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.

3.7 Testing the Transformer

Wireless Edition 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.

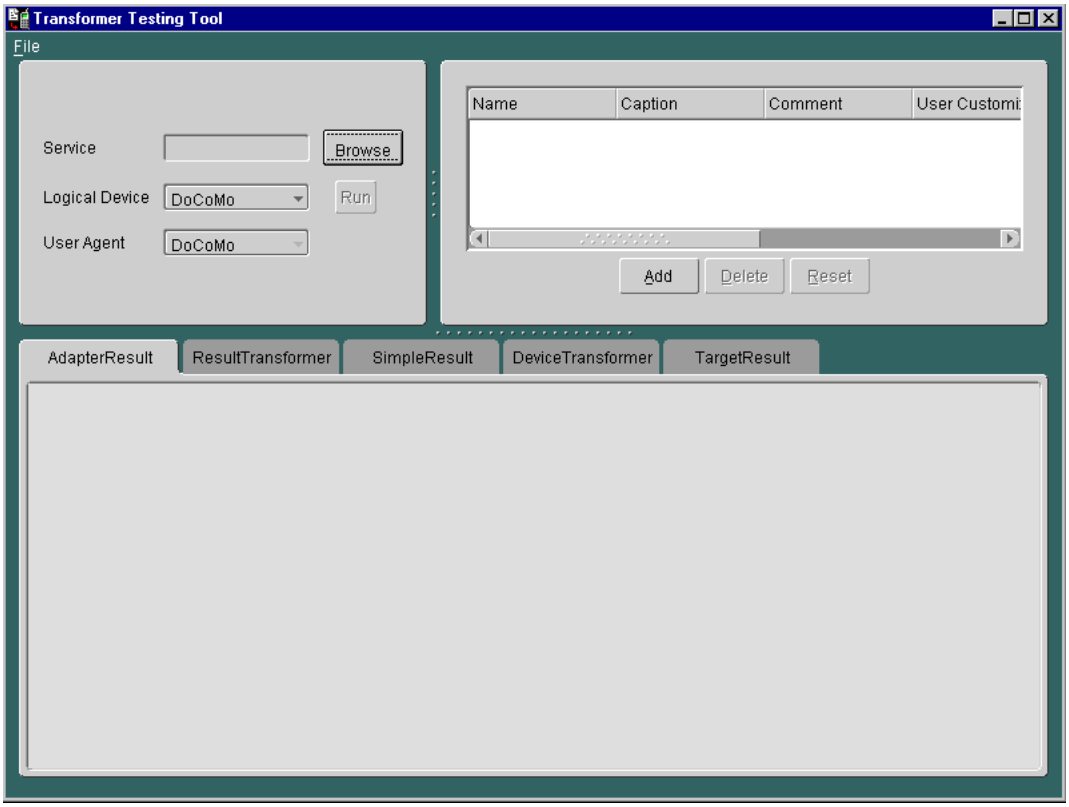
You invoke `Xslt` from the command prompt as follows:

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

3.8 Using the Transformer Testing Tool

In addition to the command-line utility, Wireless Edition enables you to test transformers using the Transformer Testing Tool. The Transformer Testing Tool allows users to test a transformer with a single tool.

Figure 3–8 The Transformer Testing Tool



The screen for this tool is divided into the following three panes:

- Services and Logical Devices
- Input Parameters
- Output Formats

Services and Logical Devices

The pane on the upper-left of the screen contains the Service, Logical Device, and User Agent fields, which you use to select the service, the logical device, and the user agent for the transformer that you want to test. To find a service, click the

Browse button to invoke a tree view of the services in the Wireless Edition repository. Click the service you want and then click OK.

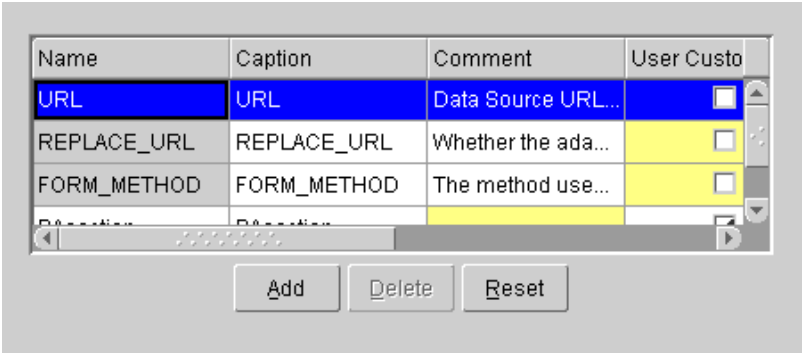
Figure 3–9 Browse Services

To select a logical device for this service, click the down arrow in the Logical Device Field to display a list of logical devices. Select a logical device from this drop-down list. You can begin testing the transformer for this master service and logical device by clicking Run. The transformer test result appears in the TargetResult panel in the Output Formats pane of the screen. You may also include input parameters as part of your testing using the Input Parameters pane.

Input Parameters

The pane on the upper right of the Transformer Testing Tool lists the input parameters for the adapter used by the master service selected in the upper-right pane.

Figure 3–10 Input Parameters



The parameters listed in this pane have the following attributes:

Table 3–3 Attributes of the Input Parameters

Parameter	Value
Name	The name of the output parameter. The Wireless Edition Service Creation Wizard sets the name of the output parameter by querying the adapter definition.
Caption	The caption is the label that Wireless Edition uses for the parameter when prompting for user input.
Comment	In the case of master services based on the Web Integration adapter, Wireless Edition automatically populates this cell with the name of the WIDL service that uses the parameter. For services based on other adapters, you can use this column to document the parameter. The comment is only used internally.
User Customizable	Specifies whether the end user can set a value for this parameter at the Personalization Portal. You can make most output 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.

Parameter	Value
Format	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.) The default format is *M. Other formats include: ■ 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. For a complete list of formats, see the *Wireless Application Protocol Wireless Markup Language Specification, Version 1.1*.
Mandatory	Select this check box if this parameter must have a value. Remove the selection for optional parameters.
Value	For most parameters, this value represents the default value for the parameter. If you specify a default value, Wireless Edition 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. The `PAsection` parameter is used by the Web Integration adapter. For `PAsection`, 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 `PAsection`, the Wireless Edition service includes all WIDL services in the WIDL interface.

You can use this panel to edit the input parameters. You can also use the Add or Delete buttons to add or remove an input parameter from the master service. Clicking Reset sets the input parameters back to their original state. After you have edited the input parameters, click Run in the upper-left pane. The test results appear in the Target Result panel in the bottom pane of the Transformer Testing Tool.

Output Formats

The bottom portion of the Transformer Testing Tool is divided into the following tabs, which show the output formats of the source content:

- AdapterResult

- ResultTransformer
- SimpleResult
- DeviceTransformer
- TargetResult

These tabs enable you to test and view the results (i.e., the AdapterResult, SimpleResult, and TargetResult) as well as edit and create the Result Transformer and Device Transformer of the service.

AdapterResult

The AdapterResult panel enables you to see the Wireless Edition source content in the AdapterResult format, the intermediary format between the source and the target output device. Source content in the AdapterResult format must be converted into SimpleResult format before it can be delivered to a target device. The AdapterResult panel is blank if no AdapterResult is produced.

ResultTransformer

The ResultTransformer panel enables you to specify a transformer that Wireless Edition uses to convert AdapterResult content to SimpleResult format.

Note: You can enable this panel and make it editable by adding and specifying a value for either the PAsection or the PAX argument in the Input Parameters panel.

SimpleResult

The SimpleResult panel shows the source content in the SimpleResult DTD format, which is the format that is needed to convert content sources from the AdapterResult into the target device format.

SimpleResult is the Wireless Edition's internal representation of the output that is returned by an adapter. If an adapter does not return its output as a SimpleResult, the master service must use the ResultTransformer to convert the AdapterResult into the SimpleResult format.

DeviceTransformer

The DeviceTransformer 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.

TargetResult

The TargetResult panel displays the source content in the format of the logical device selected in the upper-left panel.

Note: For more information on the Simple Result DTD, refer to Appendix A, "Simple Result DTD Reference".

4

Rebranding the Personalization Portal

This document explains the Personalization Portal architecture. Each section of this document presents a different topic. These sections include:

4.1 Overview

The Wireless Edition 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, XSL stylesheets, 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 restructure the framework itself by altering the logic in the JSP files and JavaBeans.

The following sections describe the elements that generate the Personalization Portal and their order of execution, as well as the file naming conventions and the directory structure used.

4.2 Page Naming Conventions

Some Personalization Portal pages display information, while others allow you to customize user information or repository object characteristics. The JSP files execute the tasks that are associated with user customization.

Each JSP file consists of an action applied to an object. One part of the JSP file name represents the action (usually Ed or Do, as in **EdFolder.jsp** and **DoFolder.jsp**). A second part of the file name represents the object (or the target) of the action, such as, Folder, Service, Alert, or Bookmark.

For example, to customize a user from the Service Subscription page **Myservice.jsp**, click the User link to invoke **EdFolder.jsp,** which displays the editable characteristics of that folder. In this example, the action is editing; it is represented by the prefix Ed in the file name. The object of the action is folder; it is represented by Folder in the file name. To apply the changes that you have made, the **DoFolder.jsp** file processes the input.

In this case, the actions are Ed and Do; the object is Folder.

- Edit JSP files, which begin with "Ed", generate the page, retrieve object characteristics, display the result, and accept user inputs.
- Action JSP files, which begin with "Do", process user inputs, perform the action, and display the result.

There are also similar JSP files at the administrator level.

- Admin JSP files, which begin with "Ad" operate like Edit JSP files, but are only available to accounts with Administrator privileges.
- Admin action JSP files, which begin with "AdDo" process inputs, perform actions and display results in the same manner as regular action files.

There are action / object combinations for all Personalization Portal objects. For example:

- **EdFolder.jsp** and **DoFolder.jsp**
- **EdService.jsp**, **DoService.jsp**, and **MyService.jsp**
- **EdBookmark.jsp** and **DoBookmark.jsp**

4.3 JavaServer Pages Structure

Each Personalization Portal JSP is composed of a series of JavaBeans assembled to generate the page when they are rendered. Each JavaBean is a reusable element and can be rendered individually or as part of the JSP.

Figure 4–1 Sample Page JavaBean Structure

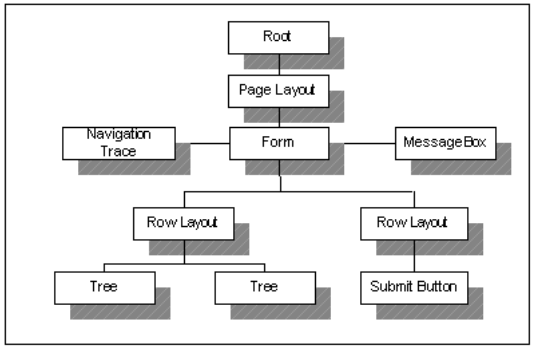


Figure 4–2 JavaBean Location on Sample Page

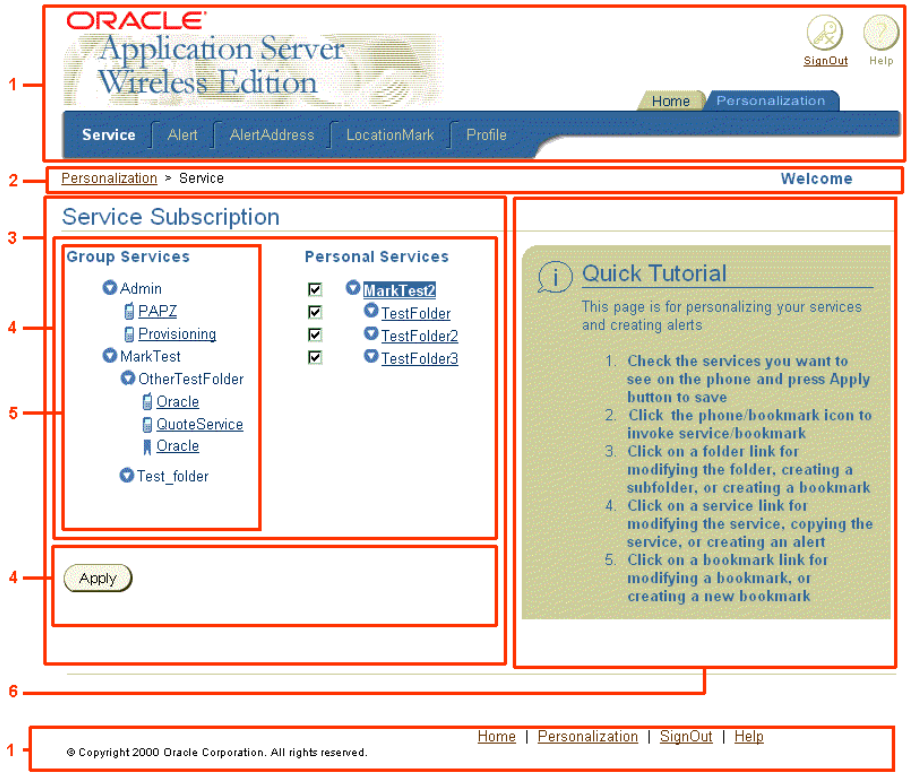


Table 4–1 JavaBean Function

JavaBean	Description
Root	Encompasses the JSP and allows it to be rendered as a whole.
1. Page Layout	Establishes the Header and Footer and reserves the remainder of the page for other content. This component contains the Navigation Trace, Form, and Message Box elements.
2. Navigation Trace	Displays navigation cue and display name elements.
3. Form	The main section of the page. This defines and contains the Row Layout.

JavaBean	Description
4. Row Layout	Contains Tree, Cell, and Button elements.
5. Tree	Linked hierarchical lists.
6. Message Box	Displays text elements.
Button	Used to approve or cancel page actions. This is part of a row layout.

Pages can be modified in one of two ways.

- You can paste a partial page into the current layout. For example, you can remove a message box by deleting that JavaBean, or replace it with a Text Box.
- You can reuse an existing JavaBean to introduce a new element on a custom page. For example, using the Tree JavaBean to place a Tree on a new page.

4.3.1 Directory Structure

To rebrand the Personalization Portal, you modify the JSP files that generate the Personalization Portal. After installing the Wireless Edition, these files are located in the *Oracle_Home*/panama/portal directory which has the following structure:

Table 4–2 Portal Directory Contents

Directory	Contents
portal	Container JSP files. Container files are accessed directly by browsers. They invoke JavaBeans and other JSP files. This directory also contains JavaScript files.
portal/_pages/_portal	The folder where the compiled pages are saved.
portal/images	Images used throughout the Personalization Portal.
portal/Web-inf/jsp	Module JSP files. These files are included by either container JSP files or other module JSP files.
portal/cabo	JavaBean stylesheet, image, and JavaScript.
portal/cabo/images	JavaBean static images.
portal/cabo/images/cache	JavaBean generated images.
portal/cabo/Libs	Javascript.
portal/cabo/styles	Stylesheets.

Directory	Contents
portal / cabo / styles / cache	Generated stylesheets.
portal / oracle / panama / webui	UI beans and **LocalStrings.properties** file

4.4 Customization Levels

Personalization Portal pages can be customized in several different ways. You can easily alter the appearance of logos, banners, and icons. Alternatively, you may want to create your own JSP to achieve the desired look and feel.

4.4.1 Appearance Customization

This method requires replacing static strings in the **HookFunc.jsp** file located in the *Oracle_Home* / panama / portal / WEB-INF / jsp directory. By changing the file names called in by these static strings, you can alter the banner art, logo art, and tool tip text.

Table 4–3 HookFunc.jsp String Usage

String	Page Element
logoImage	Page logo image
logoDesc	Page logo tool tip text
advImage	Optional advertising banner image
advDescrip	Optional advertising banner image tool tip text
advDest	Optional advertising banner image destination

4.4.2 JSP Modification

The JSP file **PageTemp.jsp** generates the Personalization Portal page template. **PageTemp.jsp** is included in other JSP files which generate different contents in each page. The following illustrates the files that generate the Personalization Portal home page.

Figure 4–3 JSP Placement in Page Structure

PageTemp.jsp generates the logo and Sign Out button at the top of the page. **MyService.jsp** presents a tree view of the services available to the user.

4.4.3 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, Wireless Edition first invokes **EdFolder.jsp**, then **DoFolder.jsp**.

Users can customize or configure the following:

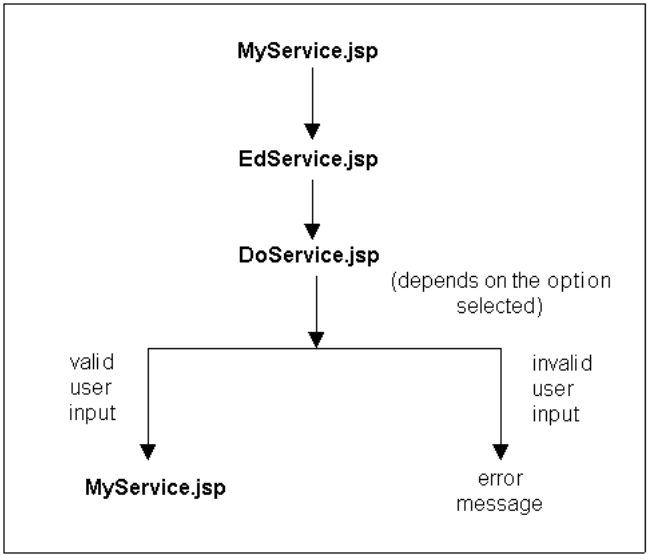
- Services

- Folders
- Alerts (alert users when preset conditions are met)
- Profile Information
- Bookmarks
- Alert Addresses
- Locationmarks

4.4.4 Flow Example - Customizing a Services

The **MyService.jsp** file displays the Service Subscription page, which allows the user to customize or copy a service. Depending on the option selected by the user, an input form is displayed by an **EdService.jsp** file. When the user clicks a button on the input form, the flow moves to a **DoService.jsp** file. The **DoService.jsp** file posts and processes input values, and, depending on the validity of the input values, returns to the **MyService.jsp** or displays an error message. The following figure displays the flow of control when a user edits a service.

Figure 4–4 Service Customization Flow



Note: The action pages, **DoService.jsp** and **DoFolder.jsp** have no display component, they perform their function, update the database, and then return the user to an input page, such as **EdFolder.jsp** or **MyService.jsp**.

4.4.5 Creating New JSP

To create a new JSP, you implement the Personalization Portal API. The classes in this API enable you to customize a version of the Personalization Portal by providing a set of interfaces for portal customization. For detailed information on the interfaces in this API, see Chapter 5, "Using the Personalization Portal API".

4.5 Setting the Multi-Byte Encoding for the Personalization Portal

The Personalization 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 JTF-8, which can handle Western European languages as well as some Asian languages. The portal sets the content for each page with the encoding specified by the logical device. To change the default encoding, click PAPZ under Logical Devices in the Service Designer and change the encoding according to the IANA standards for your particular language.

5

Using the Personalization Portal API

This document describes the Wireless Edition Personalization Portal API. Each section of this document presents a different topic. These sections include:

- Section 5.1, "Overview"
- Section 5.2, "Personalization Portal API Classes"
- Section 5.3, "Session Flow"

5.1 Overview

The Personalization Portal API classes are designed to allow you to customize a version of the Wireless Edition Personalization Portal. They provide a streamlined set of classes for portal customization. There are also built in syntax checks to verify data model logic.

The use of the Personalization Portal is a sequence of Hypertext Transaction Protocol (HTTP) requests. Each request begins with the user selecting a JavaServer Page (JSP) link which issues the request. The user enters data and clicks a button which causes the JSP to retrieve, update, or create a new repository object. Each request is controlled by classes which have been grouped into controllers based around the type of operation being performed.

Note: For detailed information regarding the Personalization Portal API, see the *Oracle9i Application Server Wireless Edition Javadoc*.

5.2 Personalization Portal API Classes

The Wireless Edition Personalization Portal is represented by a default set of JavaServer Pages (JSP) which allow you to personalize folder, service, bookmark, alert, alert address, locationmark, and profile repository objects. The Personalization Portal API enables you to create your own JSPs. The classes are categorized by specific function. Combining these functions is one method of creating your own JSP framework. See Chapter 4, "Rebranding the Personalization Portal" for other methods of creating your own JSPs.

The API is categorized into the following controllers. Each of them is stateless and can be used as a Java singleton class. Calls to these classes are not context sensitive.

5.2.1 Login and Initialize Session - RequestController

RequestController handles operations related to requests and session control such as:

- User login.
- Session creation.
- Requesting a service.
- Logging out.

Table 5–1 Request Controller API Examples

Operation	API Method Called
Login and user authentication at login page	login
Initialize session and load proper JSP at the top of each page	initRequest
Request a service	runService
Log out of Personalization Portal	logout

5.2.2 Group Creation and Modification - GroupController

GroupController is used to handle group related operations such as:

- Creating or deleting group objects.
- Retrieving and updating group properties.
- Listing the users within a group.

- Moving users in or out of a group.

Table 5–2 Group Controller API Examples

Operation	API Method Called
Group creation	createNewGroup
Display an array of all existing groups	getAllGroups
Modify group properties, such as name	setParameters
Set group root services or folders	setRootServiceHash
View and customize group owned services	getGroupService
Query and view all the group users	getGroupUserList
Delete group	deleteGroup

5.2.3 User Creation and Modification - UserController

UserController oversees operations involving users, such as:

- Create user home folder.
- Search for and display a user list by user name and / or display name.
- View and modify user profile.
- Delete single / multiple users.

Table 5–3 User Controller API Examples

Operation	API Method Called
Create a user home folder	createNewUser
Search the user list by user name or display name	QueryUserList
Display the user profile for modification	getCurrentLoginUser getGroupRootFolders getAlertAddresses getAllAlerts setDefaultLocationMark getAvailableLanguages getAvailableCountries

Operation	API Method Called
Display the services for a user's group(s)	getUserRootFolder
Deletes user(s)	deleteUser
Moves user(s)	setUserParameters

5.2.4 Object Customization -- ServiceController

ServiceController regulates operations for service, folder, and bookmark objects such as:

- Viewing the service tree.
- Create sub-folder or bookmark.
- Copying a service.
- Reordering objects within a folder.
- View a bookmark URL.
- Set object parameters.
- Setting objects as visible / invisible.
- Delete an object.

Table 5–4 Service Controller API Examples

Operation	API Method Called
Display the service tree	getChildren
Create new folder or bookmark	createNewFolder createNewBookmark
Copy a service	copyService
View the URL target of a bookmark	getExternalURL
Reorder the contents of a folder	setParameters
Make object visible or invisible	setVisibilityHash
Delete an object	deleteService

5.2.5 Alert Customization -- AlertController

AlertController handles operations of alerts and alert address settings such as:

- Create an alert or alert address.
- Retrieving the parameters of an alert.
- Setting the parameters of an alert.
- Setting the parameters of an alert address.
- Delete an alert address.

Table 5–5 Alert Controller API Examples

Operation	API Method Called
Create a new alert or alert address	createNewAlert
	createNewAlertAddress
View the parameters of an alert	getAlertFreqType
	getAlertFreqDay
	getAlertFreqValue
	getAlertFreqUnit
	getAlertStartTime
Set the parameters of an alert	setParameters
	setTimeParameters
Set the parameters of an alert address	setAddressParameters
Delete an AlertAddress	deleteAlertAddress

5.2.6 Locationmark Creation and Modification -- LocationMarkController

LocationMarkController manages operations with locationmarks including:

- Create a locationmark.
- View all the locationmarks.
- Modify a locationmark.
- Set default locationmark.
- Delete a locationmark.

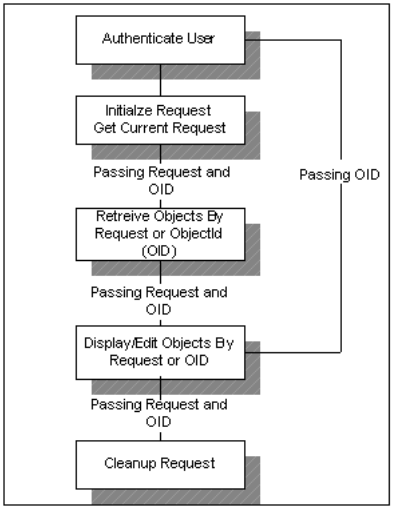
Table 5–6 LocationMarkController API Examples

Operation	API Method Called
Creating a location mark	createNewLocationMark
Display location mark parameters	getParametersHash
Set location mark parameters	setParameters
Delete a location mark	deletLocationMark

5.3 Session Flow

The following diagram displays the flow of operations in an HTTP request. RequestController handles authenticating the user and initializing the request. Other controllers are called depending on the parameters of objectId, the current request, and any input from the user in the form of input strings.

Figure 5–1 Session Flow



5.3.1 Sample Code

The following code samples demonstrate an example of an HTTP session based on the process illustrated in Figure 5–1.

5.3.1.1 Authenticate User

```
try {
     // request is HttpServletRequest
     // -1 means the application Session will never expire
     // until the HttpSession expires
     RequestController.getInstance().login(request, -1);
} catch (PortalException pe) {
}
```

5.3.1.2 Initialize Session

```
try
     // request is HttpServeltRequest
     Request _mRequest =
RequestController.getInstance().initRequest(request);
}  catch (PortalException pe) {
}
```

5.3.1.3 Retrieve Objects

```
try {
     //_mRequest is oracle.panama.rt.Request
     //currentUser is current user with type of //oracle.panama.model.User
     //currentService is current service in Request with type of
     //oracle.panama.model.Service
     User currentUser = UserController.getInstance().getCurrentUser(_mRequest);
     Service currentService =
     ServiceController.getInstance().getCurrentService(_mRequest);
} catch (PortalException pe) {
}
```

5.3.1.4 Display/Edit Objects

```
try {
     //inputHash is assigned with a hashtable of
     //inputArgument name-value pairs
     Hashtable inputHash = ServiceController.getInstance().
     getInputArguments(currentService.getId());

     //display inputArgument name and value,
     //and modify some of the values in inputHash
     ..

     //update the inputArgument values of the current service
     ServiceController.getInstance().
```

```
    setInputArguments(currentService.getId(), inputHash);
} catch (PortalException pe) {
}
```

5.3.1.5 Cleanup Request

```
//_mRequest is oracle.panama.rt.Request
RequestController.getInstance().freeRequest(_mRequest);
```

6

Using the Runtime API

This document describes the Wireless Edition Runtime API. Each section of this document presents a different topic. These sections include:

6.1 Overview

The Wireless Edition Runtime processes requests from Hypertext Transaction Protocol (HTTP) user agents and autonomous runtime agents, and invokes the services in the repository for these agents. It performs automatic session tracking and terminates the sessions when they expire after the maximum interval of inactivity or when the sessions are invalidated when the users log out from the Wireless Edition. During the request execution, the Runtime dispatches the authentication, authorization, device identification, location acquisition, data logging and other processing logics to the respective modules.

The Wireless Edition Runtime API provides a Java interface to examine the runtime execution status, monitor the runtime execution behavior, and augment the default execution semantics. The Runtime API consists of three Java packages:

1. `oracle.panama.rt` provides the interfaces to the essential Runtime objects for status examination.
2. `oracle.panama.rt.event` provides the interface to monitor the runtime execution sequence based on the Java event model.
3. `oracle.panama.rt.hook` provides the interfaces for the essential Runtime customizable components and the default implementation for these interfaces.

These three packages are included in the panana.zip file. Make sure you have included **panama.zip** in your Java classpath when you compile your Java application or plug-in modules that depend on the Runtime API.

6.2 Runtime Core

The `oracle.panama.rt` package defines the core of the Runtime API. Adapters that conform to the Runtime API must implement the `oracle.panama.adapter.RuntimeAdapter` interface. The classes that implement the RuntimeAdapter interface can use the Request, Response, Session, and ServiceContext interfaces in the `oracle.panama.rt` package.

The following sections describe the interfaces and classes in this package. The interfaces are:

- Request
- Response
- Session
- ServiceContext
- ManagedContext

The class in this package is:

- RequestFactory

6.2.1 Request

A request object is used to invoke services. Generally, it defines which service to invoke and the particular parameters needed to invoke that service. It also defines the user, device, and other Runtime contexts.

A listener can subscribe to events from a request.

The following methods in the Request interface allow you to access, replace, add, or remove the parameters that are associated with the request object:

```
Object getAttribute(AttributeCategory category, String name)

Object setAttribute(AttributeCategory category, String name,
   Object attribute)
```

The methods access the name and value of the attributes, which can be user parameters, system parameters, or the contexts for adapters, hooks, and listeners.

There are three categories of attributes:

- PARAMETERS
- RUNTIME
- CONTEXTS

The most important attribute category for Request is PARAMETERS, which contains the query parameters submitted by the user. For HTTP user agents, Runtime parses the URL query string to retrieve the parameters. The runtime agents or other internal clients can set these parameters programmatically. Since Runtime may cache and rewrite the URL for HTTP user agents, some of the parameters are maintained in the URL cache for the user. Runtime may have to parse both the query string from the HTTP request and the URL cache to build a complete list of query parameters.

The reference model, which is described in Section 6.5, "The Runtime Execution Reference Model", shows that each time a new request object is created, Runtime passes the request object to the ListenerRegistrationHook to let the hook register listeners.

Table 6–1 describes the names of the system-defined parameters which are part of the PARAMETERS AttributeCategory in Request. The left column in the table shows the Java constants that you can use to retrieve the value of the parameter from the request object.

The SimpleResults and AdapterResults can refer to the Runtime variables, which are composed from the names of the parameters in Table 6–1 by appending *two* underscore characters (__) before and after the parameter name. These Runtime variables in the SimpleResults and AdapterResults are "place holders" which are replaced by the values of the parameters during the post processing phase before the FinalResult is returned to the requester.

Table 6–1 System Defined Request Parameters

Java Program Constants Representing the Name of the Parameter in the Request Object	The Name of the Parameter in the Request Object	Description
Request.USER_NAME	"PAuserid"	The name of the user signing on to the Wireless Edition.
Request.PASSWORD	"PApassword"	The password for the user signing on to the Wireless Edition.
Request.EFFECTIVE_USER_NAME	"PAeffuserid"	The name of the effective user.
Request.SERVICE_OID	"PAoid"	The object id of the requested service.
Request.SERVICE_PATH	"PAservicepath"	The path of the requested service in the repository.
Request.SECTION	"PAsection"	The name of the result transformer in the Master Service.
Request.SESSION_ID	"PAsid"	The session id for tracking user sessions.
Request.REQUEST_LANDMARK	"PArlmk"	The landmark setting for the current request.
Request.SESSION_LANDMARK	"PAslmk"	The landmark setting for the current session.
Request.LOGOFF	"PAlogoff"	The request to log off and invalidate the session.
Request.LOGIN	"PAlogin"	The login page request.
Request.GET_PAGE	"PAgetPage"	Get the static page in the Master Service.

Line 4 in the following code example shows how the value of the PArlmk parameter can be retrieved from the Request object. Line 5 shows a statement for setting the Request parameter.

Example:

```
1. public void invoke(ServiceContext sc) {
2.    .
```

```
3.    Request request = sc.getRequest();
4.    String value = request.getParameter(Request.REQUEST_LANDMARK);
5.    request.setParameter(Request.SESSION_LANDMARK, “Redwood City”);
6.    .
7.}
```

6.2.2 Response

This interface represents the Response objects in the Runtime. A listener can subscribe to events from a Response. The Response object is the execution result of the prior Request object.

6.2.3 Session

This interface represents the session objects in the Runtime. A valid session is established after a Wireless Edition user has successfully logged in. Any request (or service invocation) can only be executed in a valid session context. A session can either expire after the session exceeds the max interval of inactivity or get invalidated when the user requests an explicit log out. Developers can store the session-long information in the corresponding session object.

A listener can subscribe to events from a session.

The reference model, which is described in Section 6.5, "The Runtime Execution Reference Model", shows that each time a new session object is created, Runtime passes the session object to the ListenerRegistrationHook to let the hook register listeners.

6.2.4 ServiceContext

A ServiceContext provides the service request context for a valid and authorized request. Semantically, a new ServiceContext object is created for each validated request. The ServiceContext stores the input parameters, output parameters, AdapterResult (if returned by the Adapter), and SimpleResult (either returned by the Adapter or generated by the ResultTransformer). The associated request and session can be accessed by way of the ServiceContext object.

The AdapterResult and SimpleResult can refer to the system defined ServiceContext parameters using the Runtime variables as "place holders," which are substituted with values during the post processing phase before the FinalResult is returned to the requester. For more information on post processing, see the discussion regarding the `PostProcessor` in Section 6.5.1, "The Reference Model".

Runtime variables are composed from the names of the parameters in Table 6–1 and Table 6–2 by appending two underscores (__) before and after the parameter name.

Example:

- SimpleResult can contain Runtime variables as follows:

  ```
  target="___REQUEST_NAME__?___SESSION__&amp; PAoid=__PAoid__"
  ```

- Given the variables above and if the following three conditions exist:
  - the value of _REQUEST_NAME is "/ptg/rm"
  - the value of _SESSION is PAsid=ukAj6hH
  - the value of PAoid is 254

  Then after the substitution of the Runtime variables, the result becomes:

  ```
  target="/ptg/rm?PAsid=ukAj6hH&amp; PAoid=254"
  ```

All the input parameters, output parameters, AdapterResult, and SimpleResult in the ServiceContext are externalized as an XML document.

This XML document is the input document for the transformers. The XSL stylesheets for the transformers must be written against the DTD for the ServiceContext's XML document.

Table 6–2 describes the system-defined ServiceContext parameters which are found among the ServiceContext arguments. The left column in the table shows the Java program constants that represent the names of the parameters in the ServiceContext object.

Table 6–2 The System Defined ServiceContext Parameters

Java Program Constants Representing the Name of the Parameter in the ServiceContext	The Name of the Parameter in the ServiceContext Object	Description
ServiceContext.DEVICE	"_LOGICAL_DEVICE'	The name of the device model.
ServiceContext.REQUEST_NAME	"_REQUEST_NAME"	The URI of the servlet. For example, if the URL is http://www.oracle.com/ptg/rm? PAoid=100 Then the URI of the servlet is: /ptg/rm

Java Program Constants Representing the Name of the Parameter in the ServiceContext	The Name of the Parameter in the ServiceContext Object	Description
ServiceContext.SESSION	"_SESSION"	The SessionId URL For example, PAsid=ukAj6hH
ServiceContext.INP_FIRST_SERVICE_URL	"_SERVICE_URL"	The URL of the requested service in the repository For example, /users/smith/news
ServiceContext.INP_FIRST_SERVICE_NAME	"_SERVICE_NAME"	The name of the requested service in the repository For example, news
ServiceContext.FIRST_ACCEPT_LANG	"_FirstAcceptLanguage"	The first language in the list of accepted languages.
ServiceContext.USER	"_User"	The effective user, which may be the authenticated user.
ServiceContext.USER_LANGUAGE	"_UserLanguage"	The user's preferred language; it should be one of the user agent's accepted languages.
ServiceContext.LONGITUDE	"_Longitude"	The current longitude location.
ServiceContext.LATITUDE	"_Latitude"	The current latitude location.
ServiceContext.SCREEN_COLS	"_ScreenColumns"	The number of columns that are displayed.
ServiceContext.SCREEN_ROWS	"_ScreenRows"	The number of rows that are displayed.
ServiceContext.SCREEN_WIDTH	"_ScreenWidth"	The width of the display.
ServiceContext.SCREEN_HEIGHT	"_ScreenHeigth"	The height of the display.
ServiceContext.USER_AGENT	"User-Agent"	The type of the user agent that is obtained from the HTTP header.
ServiceContext.ACCEPT_LANG	"Accept-Language"	The list of the languages that are accepted by the user agent.

In the following code fragment example, line 5 shows that the Java program constants can be used to refer to the parameters. Line 6 shows that the name of the

parameter can be spelled out (case sensitive). The parameter "Accept_encoding" is not one of the parameters in Table 6–1 or Table 6–2. Line 7 shows that the parameters from the request object are also available among the ServiceContext arguments. However, the ServiceContext parameters in Table 6–2 are not part of the PARAMETERS attribute category in Request objects, and they are not accessible from the Request objects.

Example:

```
1. public void invoke(ServiceContext sc) {
2.    .
3.    Arguments args = sc.getInputArguments();
4.    .
5.    String language = args.getInputValue(ServiceContext.USER_LANGUAGE);
6.    String encoding = args.getInputValue("Accept_encoding");
7.    String landmark = args.getInputValue(Request.REQUEST_LANDMARK);
8.}
```

The Java program constants in Table 6–3 below represent the names of the tags in the XML documents for the ServiceContext. The “ServiceRequest” tag is the root element of the ServiceContext. The “Result” tag contains the AdapterResult and SimpleResult. The “Arguments” tag is a sibling of the “Result” tag; it contains all input and output arguments.

Table 6–3 The XML Tag Names for ServiceContext and Results

Java Program Constants Representing the Names of the XML Tags in the ServiceContext	The Name of the XML Tag	Description
ServiceContext.SERVICE_REQUEST	"ServiceRequest"	XML element containing service context
ServiceContext.RESULT	"Result"	XML element containing adapter and simple results

The following example of the XML document for a ServiceContext shows the “ServiceRequest” tag as the root element of the ServiceContext. Some of the input arguments in the following ServiceContext example are described in Table 6–1 and Table 6–2. Several of these input arguments (tags 21 to 28) are obtained from the HTTP header attributes.

Example of the XML document for a ServiceContext:

```
1. <ServiceRequest>
       a. <Arguments>
              i. <Inputs>
                   1. <PAsid type="SingleLine"
                       usage="true">BVlcv</PAsid>
                   2. <PAoid type="SingleLine"
                       usage="true">244</PAoid>
                   3. <PAservlet type="SingleLine"
                       usage="true">rm</PAservlet>
                   4. <PAdebug type="SingleLine"
                       usage="true">1</PAdebug>
                   5. <_SERVICE_NAME type="SingleLine"
                       usage="true">Employee</_SERVICE_NAME>
                   6. <_SERVICE_NAME_ENC type="SingleLine"usage="true">
                       Employees</_SERVICE_NAME_ENC>
                   7. <_SERVICE_URL type="SingleLine" usage="true">
                       /home/Employees</_SERVICE_URL>
                   8. <_SERVICE_URL_ENC type="SingleLine" usage="true">
                       /home/Employees</_SERVICE_URL_ENC>
                   9. <_LOGICAL_DEVICE type="SingleLine" usage="true">HTML
                       </_LOGICAL_DEVICE>
                  10. <_SESSION type="SingleLine" usage="true">PAsid=BVlcv
                       </_SESSION>
                  11. <_REQUEST_NAME type="SingleLine" usage="true">/p2g/rm
                       </_REQUEST_NAME>
                  12. <_ScreenColumns type="SingleLine" usage="true">0
                       </_ScreenColumns>
                  13. <_ScreenRows type="SingleLine" usage="true">0
                       </_ScreenRows>
                  14. <_ScreenWidth type="SingleLine" usage="true">0
                       </_ScreenWidth>
                  15. <_ScreenHeigth type="SingleLine" usage="true">0
                       </_ScreenHeigth>
                  16. <_User type="SingleLine"
                       usage="true">user1</_User>
                  17. <_UserLanguage type="SingleLine"
                       usage="true"/>
                  18. <_FirstAcceptLanguage type="SingleLine" usage="true">ja
                       </_FirstAcceptLanguage>
                  19. <_Longitude type="SingleLine"
                       usage="true"/>
                  20. <_Latitude type="SingleLine"
                       usage="true"/>
```

```
                    21. <accept type="SingleLine" usage="true">image/gif,
                           image/x-xbitmap, image/jpeg,
                        image/pjpeg, image/png, */*</accept>
                    22. <accept-charset type="SingleLine" usage="true">
                        iso-8859-1,*,utf-8</accept-charset>
                    23. <accept-encoding type="SingleLine"
                        usage="true">gzip</accept-encoding>
                    24. <host type="SingleLine"
                        usage="true">localhost</host>
                    25. <cookie type="SingleLine" usage="true">
                           kurt=NTJCMUIzNzczQTA1QzBFRDAxNzY
                        3ODdBNEYxNTc0RkYwMDc1Rjc1MjFFU29ubnk=</cookie>
                    26. <accept-language type="SingleLine"
                        usage="true">ja,en</accept-language>
                    27. <connection type="SingleLine"
                        usage="true">Keep-Alive</connection>
                    28. <user-agent type="SingleLine"
                           usage="true">Mozilla/4.5
                        [en] (WinNT; U)</user-agent>
          ii. </Inputs>
   b. </Arguments>
   c. <Result>
           i. <AdapterResult>
               1. <Table>
                    a. <Row>
                         i. <EMPNO>10</EMPNO>
                        ii. <FNAME>Scott</FNAME>
                    b. </Row>
                    c. <Row>
                         i. <EMPNO>20</EMPNO>
                        ii. <FNAME>Tiger</FNAME>
                    d. </Row>
               2. </Table>
          ii. </AdapterResult>
         iii. <SimpleResult>
               1. <SimpleContainer name="Services">
                    a. <SimpleMenu name="alias" title="Employees">
                        b. <SimpleMenuItem
                                 target="/p2g/rm?PAsid=BVlcv&#38;
                             PAckey=6!">Scott</SimpleMenuItem>
                        c. <SimpleMenuItem
                                 target="/p2g/rm?PAsid=BVlcv&#38;
                             PAckey=7!">Tiger</SimpleMenuItem>
                    d. </SimpleMenu>
               2. </SimpleContainer>
```

```
            iv. </SimpleResult>
        d. </Result>

2.</ServiceRequest>
```

6.2.5 ManagedContext

In many situations, the customized hooks, listeners, and adapters require session-long, user-defined context information to be stored in the session object, so that subsequent calls or requests can access the context information. Furthermore, these application contexts may contain system resources that should be freed when the session is closed.

The user-defined context must implement the ManagedContext interface and provide customized implementation for the `invalidate` method. The customized hooks, listeners, and adapters can register the session-long application context object with the session through the `setManagedContext` method. The `invalidate` method will be called by Runtime when the session terminates.

6.2.6 RequestFactory

The `RequestFactory` class is defined in the `oracle.panama.rt` package. The `RequestFactory` provides the APIs to programmatically create request objects to be executed. The `RequestFactory` creates the request objects that, when executed, initiate the runtime controllers to process the service requests by invoking the necessary business processes, such as session management, authentication, authorization, service invocation, and result transformation.

Case 1: Application of the RequestFactory Pattern in the HTTP Servlet

This case uses the `ParmImpl` servlet to illustrate the `RequestFactory` pattern. The following code example is the `doGet()` method of the `ParmImpl` servlet:

```
1. public void doGet(HttpServletRequest request, HttpServletResponse response)
2.            throws IOException, ServletException {
3.      Request req = RequestFactory.createRequest(request, response);
4.      if (req == null) {
5.            return;
6.      }
7.      try {
8.            Response resp = req.execute();
9.      } catch (Exception ex) {
10.     } finally {
11.           req.invalidate();
12.     }
13.}
```

Line 3 in the above example illustrates the use of the static method `createRequest(HttpServletRequest request, HttpServletResponse response)` of `RequestFactory` to create a `Request` object.

When the `Request` object is executed in line 8, it returns a `Response` object.

The Java code in the above example does not include reading or writing of the content in the `Response` object because the runtime controller directly transfers the content to the HttpServletResponse object.

The `execute()` method of the `Request` object starts a control flow which performs the following sequence of processes:

1. Assign a session to the request.
2. Parse the URL parameters in the HttpServletRequest.
3. Authenticate the user if the user credentials are provided among the parameters.
4. Authorize the requested service.
5. Invoke the service.
6. Transform the XML result from the service invocation.
7. Convert the final XML result to a string.
8. Set the response string in the HttpServletResponse.
9. Return to the servlet.

Line 11 in the code example invalidates the completed object, thereby freeing all the resources associated with the request object.

Case 2: Application of the RequestFactory Pattern in the Runtime Agent

The following case illustrates how a runtime agent uses the `RequestFactory` pattern to request services from the Wireless Edition Runtime.

```
1.import oracle.panama.rt.RequestFactory;
2.import oracle.panama.rt.Request;
3.import oracle.panama.rt.Response;
4.import oracle.panama.rt.Session;
5.import oracle.panama.rt.ServiceContext;
6..
7.import oracle.panama.model.MetaLocator;
8.import oracle.panama.model.ModelServices;
9.import oracle.panama.model.Service;
10.import oracle.panama.model.User;
11.import oracle.panama.model.AlertAddress;
12..
13..
14..

15.Session signon(String user, String password) throws PanamaException {
16.     Request request = RequestFactory.createRequest(user, password);
17.     request.validate();
18.     Session session = request.getSession();
19.     request.invalidate();
20.     return session;
21.}

22.String invokeService(Session session, Service service, User user,
                        AlertAddress address, String symbol) {
23.     Request req;
24.     Response resp;
25.     ServiceContext sc;
26.     String content = null;

27.     try {
28.          req = RequestFactory.createRequest(session, service,
                                                 user, address);
29.          if (req == null) {
30.               return null;
31.          }
32.          try {
33.               req.setParameter("TickerSymbol", symbol);
34.               sc = req.validate();
```

```
35.                  resp = req.execute();
36.                  if (sc.isAnyResultPresent()) {
37.                      content = resp.getContent();
38.                  }
39.              } catch (Exception ex) {
40.              } finally {
41.                  req.invalidate();
42.              }
43.              return content;
44.         }
45. }

46.String userName;
47.String password;
48.String effectiveUserName;
49.String symbols[];
50.
51.void main() {
52.     ModelServices models = MetaLocator.getInstance().getModelServices();
53.     User user = models.lookupUser(effectiveUserName);
54.     Service service = models.lookupService("YahooQuote");
55.     AlertAddress[] addresses = user.getAddresses();
56.     Session session = signon(userName, password);
57.     for (int i = 0; i < symbols.length(); i++) {
58.         .
59.         .
60.         String content = invokeService(session, service, user, addresses[0],
                                symbol[i]);
61.         .
62.         .
63.     }
64. }
```

The `signon()` method signs on the user to the Runtime. When the `Request` object is validated in line 17, the user name and password credentials are used to authenticate the user. Since no service is invoked during the sign-on request, the code example shows that the `Request` object is not executed.

If there is no exception after validation, the authenticated session is retrieved from the `Request` object in line 18.

The `Session` object is used in the `invokeService()` method for subsequent requests to the Runtime. Line 28 in the `invokeService()` method creates a `Request` object for an effective user and a specified service. For this operation to

succeed, the authenticated user must have administrative privileges over the effective user account.

The address parameter identifies the target device model for the Runtime to format the content in the appropriate markup language.

The main routine in the above code example illustrates how it iteratively invokes the service each time with a different input parameter. The contents returned by each service request can be combined into a larger document and sent to the user.

6.3 Event, Listener

During the establishing of a session, the expiration of a session, or the processing of a request, Runtime can generate a sequence of events to signal the execution progress if any interested listener is registered with these objects. Generally, listeners should not be intrusive to the Runtime. They should monitor the Runtime progress instead of altering its execution behavior. The possible applications for the event package can be a logger, a billing procedure, or a performance monitor tool. The `oracle.panama.rt.event` package defines the Listener and Event API.

Listeners listen to Events. Listener and Event form an important design pattern in which the Listener is an observer. Three types of listeners are defined:

- RequestListener
- SessionListener
- ResponseListener

The registration of a listener by the ListenerRegistrationHook conforms to the publish-subscribe model. The ListenerRegistrationHook subscribes the listeners to receive events from the subject, such as Request, Response, or Session.

6.3.1 Implementing the RequestListener Interface

The implementation of `oracle.panama.rt.event.RequestListener` can receive the Request-related event.

The possible Request-related events can include any of the following events:

- `before request`
- `request begin`
- `request end`
- `service begin`

- `service end`
- `transform begin`
- `transform end`
- `after request`
- `request error`

The timing sequence regarding when the event is generated is discussed in Section 6.5, "The Runtime Execution Reference Model". However, not all the Request-related events will be generated. Which specific Request-related event will be generated is controlled by the event mask in the **System.properties** file. Note that the event mask is system-wide.

For example, if you want to generate the `request begin` event and have your `RequestListener` receive the event, you should set the `request.begin` mask to true in the System.properties file as shown below:

```
Event.request.begin=true.
```

The reference model, which is described in Section 6.5, "The Runtime Execution Reference Model", indicates that the `RequestListener` can intercept the input parameters during the `requestBeginRequest(Event)`(Step 11 in Section 6.5.1, "The Reference Model") and apply additional business rules to the request parameters before service invocation.

6.3.2 Implementing the ResponseListener Interface

The implementation of `oracle.panama.rt.event.ResponseListener` can receive the Response-related event. The only possible Response-related event is `response error`. The event constant is defined in the `oracle.panama.rt.event.ResponseEvent` interface. If you want Runtime to generate the `response error` event when the response has an error and have your `ResponseListener` receive the event, you should set the `response error` mask to true in the System.properties file as shown below:

```
Event.response.error=true
```

6.3.3 Implementing the SessionListener Interface

The implementation of `oracle.panama.rt.event.SessionListener` can receive the Session-related event. The possible Session-related events include:

- `before session`

- `session begin`
- `session end`
- `after session`

The timing sequence regarding when the event is generated is discussed in Section 6.5, "The Runtime Execution Reference Model". However not all the Session-related events will be generated. Which specific Session-related event will be generated is controlled by the event mask in the **System.properties** file. Note that the event mask is system wide.

For example, if you want to generate the `session begin` event and have your `SessionListener` receive the event, you should set the `session begin` mask to true in the System.properties file as shown below:

```
Event.session.begin=true
```

6.3.4 Guidelines

The following guidelines describe how to set up the customized Event Listener:

1. Implement the `RequestListener`, `ResponseListener`, or `SessionListener` interface.
2. Implement the `ListenerRegistrationHook` interface.
3. Compile the new Java source files from Steps 1 and 2 with the **panama.zip** file in the classpath.
4. Modify the event mask entries in the **System.properties** file to enable the generation of specific events.
5. Modify the entries to the `ListenerRegistrationHook`.
6. Modify the **class.wrapper** in the **JServ.properties** file to include the classpath for the new Java classes.
7. Restart JServ (or the Oracle HTTP Server).

Any of the event listeners may raise the AbortServiceException to signal the Runtime controller to reject the request, but this veto signal is effective only if it is raised during one of the following events when the service is yet to be invoked:

- `beforeRequest(RequestEvent)`
- `beforeSession(SessionEvent)`

- `requestBegin(RequestEvent)`
- `sessionBegin(SessionEvent)`
- `serviceBegin(RequestEvent)`

The listeners may raise the AbortServiceException during the `serviceEnd()`, `transformBegin()`, and `transformEnd()` events to refuse the service's content to the user, although any durable effect of the service invocation cannot be rolled back.

The `sessionEnd()`, `afterSession()`, `requestEnd()`, and `afterRequest()` methods should not raise the AbortServiceException.

A listener that implements the Request, Response, and Session listener interfaces is described in the code example in Section 6.8.2, "Event Listener Example". The listener in this example listens to all Request, Response, and Session events. This listener logs the response time, service time, and transform time of the requests.

The values placed in the event object persist through the life cycle of the event source and can be retrieved during subsequent events. Alternatively, the listener may place the values in the RUNTIME attribute category of the Request or Session objects. Both techniques allow the listeners to correlate and trace the events from individual event sources.

6.4 Hooks

The Runtime specifies the hook interfaces for standard plug-in modules. The following sections describe the hooks in the order in which they are applied by the Runtime. See the reference model in Section 6.5, "The Runtime Execution Reference Model".

In the Wireless Edition Runtime API, Hook and Policy form a design pattern, within which Policy is the default implementation of Hook.

The following table lists the Hooks and the default Policies that correspond to the hook interfaces:

Table 6–4 Classes that Implement the Default Policies

Hook Name	Policy Name
AuthenticationHook	AuthenticationPolicy
AuthorizationHook	AuthorizationPolicy
CallerLocationHook	CallerLocationPolicy

Hook Name	Policy Name
DeviceIdentificationHook	DeviceIdentificationPolicy
FolderRendererHook	FolderRendererPolicy
HomeFolderSorterHook	HomeFolderSorterPolicy
ListenerRegistrationHook	ListenerRegistrationPolicy
LocationServiceVisibilityHook	LocationServiceVisibilityPolicy
PostProcessorHook	
PreProcessorHook	
ServiceVisibilityHook	ServiceVisibilityPolicy
SessionIdHook	SessionIdPolicy
SignOnPagesHook	SignOnPagesPolicy
SubscriberIdHook	
SystemPasswordEncryptionHook	

6.4.1 The ListenerRegistrationHook

The reference model, which is described in Section 6.5, "The Runtime Execution Reference Model", shows that each time a new Session or Request object is created, Runtime passes the Session or Request object to the ListenerRegistrationHook to let the hook register listeners. The listener registration module can be customized to let the listeners selectively observe the event sources.

For example, a custom listener registration policy may subscribe the listeners only to the requests for the Web Integration Adapter services. Such a listener may add business rules to the Runtime controller.

6.4.2 The DeviceIdentificationHook

Runtime uses the DeviceIdentificationHook to determine the device model for the user agent. For HTTP clients, the user-agent type is the value of the “User-Agent” attribute in the HTTP header. The DeviceIdentificationHook can implement robust determination of the type of user agents for cases where the user-agent attribute is not supplied in the HTTP header.

This hook provides a mapping of the user-agent type to the device model. Runtime agents can specify the Device in the `RequestFactory` method. If the Device is specified, the Runtime controller will not invoke the DeviceIdentificationHook.

Although customization and extensions are supported, the default device identification policy is fully functional.

6.4.3 The SessionIDHook

The Wireless Edition Runtime uses the SessionIdHook to uniquely identify each new session it creates with a `Session id`. This `Session id` is used in the URLs for session tracking. It is important for custom `Session id` modules to generate long `Session id` strings. Longer `Session id` strings are less vulnerable to attack.

6.4.4 The AuthenticationHook

The Wireless Edition Runtime dispatches the authentication operations to the authentication module that implements the AuthenticationHook. The AuthenticationPolicy provides a public interface to the default authentication policy in Runtime. The default policy uses the user name and password credentials located in the Wireless Edition repository.

A different implementation of the AuthenticationHook using an external module may use any custom authentication scheme to validate the user. The external authentication module may optionally fail over to the default authentication policy.

The AuthenticationHook returns the AuthenticationContext if the authentication succeeds. Otherwise, the hook raises the AuthenticationException. The AuthenticationContext that is returned by the authentication module specifies the User object for the Session. This User object may be located in the Wireless Edition repository or provisioned by the authentication module on demand.

The AuthenticationContext is passed to the AuthorizationHook for service authorization. The `String getAuthenticationType()` method in Request can provide the name of the authentication scheme used by the plug-in authentication module, which extends the "BASIC", "DIGEST", or "SSL" authentication schemes supported by the javax.servlet.http package.

Runtime provides infrastructure support to mix and match different authentication, authorization, and provisioning policies by delegating the authentication operation to the AuthenticationHook and the authorization operation to the AuthorizationHook.

Runtime places the AuthenticationContext, which is returned by the AuthenticationHook, in the Session. The AuthenticationContext is passed only to the AuthorizationHook and is not accessible through the public interface.

The AuthenticationHook may either create the user or look up the user in the repository. If the user is provisioned by the external accounting system, the AuthenticationHook will also provision the home folder and group for the user. The user, which is returned through the AuthenticationContext, becomes the authenticated user of the session. Although large-scale customization and extension efforts are supported, the built-in authentication and authorization policies are fully functional.

6.4.5 The SignOnPagesHook

The SignOnPagesHook generates the sign-on pages for Runtime. This hook is not shown in the execution reference model because it is invoked only when the AuthenticationPolicy raises the AuthenticationFailOverException to request the sign-on pages.

When the SignOnPagesHook generates the sign-on page, the Wireless Edition Runtime sends that sign-on page to the user, who submits the user's name and password for authentication by the AuthenticationPolicy. The sign-on page is returned to the user agent only after the authentication module fails to authenticate the user by using any combination of credentials besides the user name and password.

The default policy for the SignOnPagesHook obtains the sign-on page from the device model in the repository. The default policy is fully functional without customization. The device model is identified and provided to the SignOnPagesHook. This hook returns the sign-on page formatted in the markup language for the target user agent. The hook may use the XML transformers of the device to render the sign-on page as long as the input to the transformer conforms to the SimpleResult DTD.

6.4.6 The SubscriberIDHook

Runtime invokes the SubscriberIdHook to determine the subscriber's ID.

The `SubscriberId` may be supplied by the external accounting system, by one of the fields (such as, the external ID field) of the user object in the repository, or by one of the attributes in the HTTP header. This `SubscriberId` is associated with the authenticated session.

6.4.7 The AuthorizationHook

The authentication operation is performed only one time to establish a session for the user. The authorization operation is performed for each request to the Wireless Edition Runtime.

The authorization module that implements the AuthorizationHook may use any custom authorization scheme. It is probable for the same party to implement both the AuthenticationHook and the AuthorizationHook. For example, in an environment that uses a pre-billing scheme, the AuthenticationHook provides the AuthenticationContext that indicates the user's prepaid level or type of service to the AuthorizationHook.

The external authorization module may optionally fail over to the default authorization policy by delegating to the AuthorizationPolicy provided in the public package. The default authorization policy authorizes the service using the visibility, validity, ownership, and group membership configuration in the repository.

6.4.8 The CallerLocationHook

The CallerLocationHook provides the interface to acquire a caller's physical location in terms of latitude and longitude. The Wireless Edition provides two different default implementations of the CallerLocationHook interface.

The `oracle.panama.rt.common.CallerLocator` class provides the simple implementation using the locationmarks. The landmark object is one way of specifying the longitude and latitude position. The user can change the landmark setting in the session through the URL parameter PAslmk. If the automatic location acquisition is disabled, the landmark setting in the session supplies the current position of the mobile device to the location-based services.

The `oracle.panama.rt.hook.LocAcq` provides the automatic location acquisition implementation if the user selects the Auto-Locate option. If the automatic acquisition fails or if the user selects the manual position option, then the prior locationmark semantics will be applied.

See the oracle.panama.mp section for details on how to specify which mobile position server (either Ericsson or SignalSoft, or another customized server) is used to acquire the caller's location.

6.4.9 Service

Services are Wireless Edition repository objects. A Master Service object contains a RuntimeAdapter, which is chief among plug-in components. Folders are a type of service used for organizing other folders and services in the repository. The following three hooks control how the content of a folder gets rendered:

- FolderRendererHook
- HomeFolderSorterHook
- LocationServiceVisibilityHook

The FolderRendererHook uses the HomeFolderSorterHook and LocationServiceVisibilityHook to render the contents of the folder. When the user first signs on to the system, Runtime invokes the user's home folder. The built-in FolderRendererHook combines the contents of the home folder with the folders and services from one or more of the user's groups. The HomeFolderSorterHook is useful for sorting the group elements among the user's own elements. The LocationServiceVisibilityHook selects from the location-based subfolders in the folder for those whose regions intersect with the current position of the mobile device.

6.4.10 The PreProcessorHook, Transformer, and PostPorcessorHook

If the PreProcessorHook is specified, the Runtime invokes the PreProcessorHook to process the SimpleResult from the service invocation. The DeviceTransformer is applied to the result of the PreProcessorHook. If specified, the PostProcessorHook is invoked to process the markup page that is generated by the DeviceTransformer.

6.5 The Runtime Execution Reference Model

The following sections discuss the request factory pattern and the Runtime execution reference model for it.

6.5.1 The Reference Model

This section describes the Wireless Edition Runtime reference model, showing how the hooks and listeners participate in the processing of a service request — in this case, the request involves authentication and session establishment.

The sequence in the model shows how a service in the repository is invoked after authentication. If no service is specified in the request, as is the case for sign-on pages, the service which is invoked is that of the user's home folder.

6.5.1.1 Case: A Request Involving Session Establishment and Authentication

This is a description of the flow in how Runtime processes the events in a request that needs a new session and authentication. The numbers indicate the sequence of the actions in the Runtime.

1. `createRequest(HttpServletRequest,HttpServletResponse)`

   `ParmImpl` submits an HTTP request containing input parameters to the `RequestFactory` to create the `Request` object.

2. `registerRequestListeners(Request)`

   `Runtime` passes the object to the `ListenerRegistrationHook` to let it register listeners.

3. `beforeRequest(RequestEvent)`

   The event source `Request` issues a notification to each of the `RequestListeners`, passing the `RequestEvent` object.

4. `execute()`

   `ParmImpl` executes the newly created `Request` object, which indicates to `Runtime` that a sequence of activities will follow.

5. `findDeviceType(String)`

   `Runtime` dispatches to the DeviceIdentificationHook to determine the device model.

6. `createSessionId()`

   `Runtime` dispatches to the `SessionIdHook` to create a new session id for the `PAsid` parameter.

7. `createSession()`

   `Runtime` creates a new `Session` for the given session id.

8. `registerSessionListeners(Request,Session)`

   `Runtime` passes the new `Session` to the `ListenerRegistrationHook` to let it register the session listeners.

9. `beforeSession(SessionEvent)`

   The event source `Session` issues a notification to each of the `SessionListeners`, passing the `SessionEvent` object.

10. `parseInputParameters()`

`Runtime` parses the URL in the HTTP request and extracts the input parameters.

11. `requestBegin(RequestEvent)`

    The event source `Request` issues a notification to each of the `RequestListeners`, passing the `RequestEvent` object.

12. `authenticate(String,String,Request)`

    `Runtime` dispatches to the `AuthenticationHook` to authenticate the user.

13. `getSubscriberId(Request,Session)`

    `Runtime` dispatches to the `SubscriberIdHook` to obtain the subscriber id of the user, which can be used by the `CallerLocationHook`.

14. `sessionBegin(SessionEvent)`

    The event source `Session` issues a notification to each of the `SessionListeners`, passing the `SessionEvent` object.

15. `getCurrentLocation(Request)`

    `Runtime` dispatches to the `CallerLocationHook` to determine the location of the caller (mobile device).

16. `authorize(User,Service,Request,AutheticationContext)`

    `Runtime` dispatches to the `AuthorizationHook` to authorize the requested service.

17. `serviceBegin(RequestEvent)`

    The event source `Request` issues a notification to each of the `RequestListeners`, passing the `RequestEvent` object.

18. `invoke(ServiceContext)`

    `Runtime` invokes the service in the repository, passing the `ServiceContext` object.

19. `serviceEnd(RequestEvent)`

    The event source `Request` issues a notification to each of the `RequestListeners`, passing the `RequestEvent` object.

20. `transformBegin(RequestEvent)`

    The event source `Request` issues a notification to each of the `RequestListeners`, passing the `RequestEvent` object.

21. `process(Request,Element)`

    `Runtime` dispatches to the `PreProcessorHook` to process the SimpleResult output of the service.

22. `rewriteResultURLs(Element)`

    `Runtime` replaces the original URL with an encoded URL that contains the `PAsid` and `PAckey` parameters for the session id and the URL cache key, respectively.

23. `transform(Element,LogicalDevice)`

    `Runtime` invokes the device ResultTransformer to transform the SimpleResult to the device's markup language.

24. `process(String,Arguments,Device)`

    `Runtime` invokes the `PostProcessor` to parse the content of the device markup page. The `PostProcessor` replaces the Runtime variables (which are "place holders") with the values of the variables. For example, `"PAsid=xyzw"` replaces `___SESSION__`.

    For more information about Runtime variables, see Section 6.2.4, "ServiceContext" and Appendix B, "Runtime System Variables".

25. `process(Request,Response,String)`

    `Runtime` dispatches to the `PostProcessorHook` to process the device markup page to produce the FinalResult.

26. `transformEnd(RequestEvent)`

    The event source `Request` issues a notification to each of the `RequestListeners`, passing the `RequestEvent` object.

27. `writeContent()`

    `Runtime` writes the content to the `HTTPServletResponse`.

28. `requestEnd(RequestEvent)`

    The event source `Request` issues a notification to each of the `RequestListeners`.

29. `invalidate()`

    `ParmImpl` invalidates the `Request` object.

30. `afterRequest(RequestEvent)`

The event source `Request` issues a final notification to the `RequestListeners`, passing the `RequestEvent` object.

6.6 System Parameters

There are two different kinds of system parameters: static and derived parameters. The following sections discuss these two types of system parameters.

6.6.1 Static System Parameters

The static system parameters that are listed in Table 6–1 and Table 6–2 are usually presented in the valid ServiceContext. You can access them in one of two ways:

- Programmatically through the ServiceContext or the request object:

  `sc.getParameters(X)`

  where X is the parameter name.

- Through the Wireless Edition stylesheet PostProcessor in the FinalResult markup language as __X__ using two underscores as the prefix and two underscores as the suffix around X.

The HTTP headers sent together with the HTTP service request invocation are also considered static parameters. However which HTTP header is present depends on the browser and on the gateway. To find out which HTTP headers are present in a request, use the following:

```
Enumeration in_http_headers = Req.getHeaderAttributes()
```

This returns an enumeration of present HTTP headers in the request.

You can retrieve the HTTP header's value by enumerating the enumeration:

```
 while (in_http_headers.hasMoreElements())
 {
    String arg = (String) in_http_headers.nextElement();
    System.out.println(arg+"= "+ Reg.getParameter(arg));
}
```

6.6.2 Derived System Parameters

The second kind of system parameters is the derived parameters. A derived parameter's value is usually not present. To make its value present in the valid request object, do the following:

Add the derived parameter X to the master service and make the derived parameter X mandatory.

After each request has been validated, the Runtime computes the values for the mandatory derived parameters. Then the values of these derived parameters can be accessed in the same way as the values of the static system parameters. The Runtime-defined derived system parameters are listed in the following table:

Table 6–5 Derived System Parameters

Derived System Parameter Name	Description
_Longitude	The longitude component of the geocoding of the current requester's location
_Latitude	The latitude component of the geocoding of the current requester's location
_State	The state from which the current requester is initiating the request
_Postalcode	The postal code of the current requester's location
_Country	The country in which the current requester is initiating the request

6.7 General Guidelines on User-Defined Listeners and Hook Implementation

Component developers can develop new types of runtime agents and adapters by using only the classes and interfaces in the public packages provided in the **panama.zip** file. **The panama_core.zip** file contains the internal packages that should only be used by built-in components.

The following steps describe how you provide your own implementation of listeners and hooks.

6.7.1 Implementing the Respective Interface

The user-defined listeners and hooks should implement the respective listener interface or the hook interface. For example, if you define your own AuthenticationHook, your new AuthenticationHook Java class should implement the `oracle.panama.rt.hook.AuthenticationHook` interface.

Furthermore, the new implementation should implement the following Singleton pattern:

```
class yourClass  implement Xhook  {
     public static Xhook getInstance() {   …. }
…
}
```

6.7.2 Compile Your Java Source

Make sure you have included the **panama.zip** in your Java classpath during compilation.

6.7.3 Plug in Your Implementation through Property File

Set the corresponding entry in the **System.properties** file to specify the name of the class that provides the implementation.

The following table lists the property entry name in the **System.properties** file for each hook.

Table 6–6 Property Entry Names in the System.properties File

Hook Name	Property Name
AuthenticationHook	locator.authentication.hook.class
AuthorizationHook	locator.authorization.hook.class
CallerLocationHook	locator.caller.location.hook.class
DeviceIdentificationHook	locator.device.identification.hook.class
FolderRendererHook	locator.folder.renderer.hook.class
HomeFolderSorterHook	locator.home.folder.sorter.hook.class
ListenerRegistrationHook	locator.listener.registration.hook.class
LocationServiceVisibilityHook	locator.location.service.visibility.hook.class
PostProcessorHook	locator.post.processor.hook.class
PreProcessorHook	locator.pre.processor.hook.class
ServiceVisibilityHook	locator.service.visibility.hook.class
SessionIdHook	locator.session.id.hook.class
SignOnPagesHook	locator.signon.pages.hook.class

Hook Name	Property Name
SubscriberIdHook	locator.subscriber.id.hook.class
SystemPasswordEncryptionHook	locator.SystemPasswordEncryptionHook.class

For example, if you provide your own implementation of the authentication hook, you should set the locator.authentication.hook.class in the System.properties file to:

```
locator.authentication.hook.class = <your class name>
```

6.7.4 Modify the JServ Property File

Add a new entry of wrapper.classpath in the Jserv.properties file to include the new path in which the new Java class files reside.

6.7.5 Restart the Server

You have to restart either the Apache Server if the JServ is launched automatically by the Apache Server or the JServ if the JServ is started manually. Then, your new implementation takes effect.

6.7.6 Tips and Hints

When implementing the new listeners and hooks, consider also the following points:

6.7.6.1 Concurrent Requests

The Runtime supports concurrent instances of requests from user agents through an HTTP connection. Concurrent requests are not permitted for the runtime agent that shares the same administrator session among different effective users. For this type of agent, the Runtime serializes the requests under the same session. Concurrency is achieved by introducing more than one instance of the runtime agents, each with its own authenticated session.

6.7.6.2 Recursive Instances of Requests

The Runtime supports recursive instances of requests under the same session. Recursive instances of requests may be issued by the plug-in components, for example, to recursively invoke all services under a folder.

6.7.6.3 Query Parameters

The Runtime parses the URL query strings from HTTP user agents to retrieve query parameters. For other agents that do not use URL strings, the Runtime lets the agents set the query parameters programmatically. The Runtime allows the agents to specify the session, user, device, and service using objects instead of names.

6.7.6.4 Runtime Object References

This design constraint requires that plug-in components do not retain references to the Runtime objects across invocations.

Plug-in components may execute under asynchronous threads; in this case, the synchronous methods in the components should make snapshots of the Runtime objects before handing them to the asynchronous threads.

6.7.6.5 Thread-Safe and High-Concurrency

Since a single instance of the customized listeners and hooks is created according to the Singleton design pattern, the Java class should provide a thread-safe but very high concurrent implementation. Otherwise, the performance of the Wireless Edition Runtime can be significantly degraded.

6.8 Examples

The following examples are available in the respective subdirectories under PANAMA_HOME\sample.

6.8.1 User-Defined Hooks

The following two examples illustrate how you can develop user-defined hooks:

6.8.1.1 Example 1

This example implements a user-defined FolderRendererHook. By examining the ptg_service_log table, the MyFolderRenderer displays the content of any folder for the current user based on the number of clicks the user has made in the past.

Note especially the High-Concurrency design in this example. Also, note the use of the ManagedContext interface.

The MyFolderRenderer.java Source Code:

```
/*
 *
The MyFolderRenderer displays the content of any folder for the current
user based on the number of clicks the user has made in the past
by examining the ptg_service_log table.
*
To use:
1. configure the System.properties to enable the DB-based System logger.
2. Compile the MyFolderRenderer.java and MyFolderRendererContext.java.
Make sure you have included the panama.zip, panama_core.zip
and xmlparserv2.jar in the classpath.
3. Modify the jserv.properties to include the class files
in the wrapper.classpath.
*
Area of Improvement:
1. Instead of examining the ptg_server_log table, examine
another clickstream summary table whose data is computed offline
once aday.
2. Instead of dedicating a separate database connection for each
session for retriving the folder content's click stream summary,
use a database connection pool to avoid exhausting the db
connection resource.
*
*
*/
package hook;

import oracle.panama.rt.hook.FolderRendererHook;
import oracle.panama.rt.ServiceContext;
import oracle.panama.rt.ManagedContext;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import oracle.panama.model.Service;
import oracle.panama.model.ModelServices;
import oracle.panama.model.MetaLocator;
import oracle.panama.model.User;
import oracle.panama.model.Folder;
import oracle.panama.PAPrimitive;
```

```
import oracle.panama.core.util.Locator;
import oracle.panama.core.admin.L;

public class MyFolderRenderer implements FolderRendererHook {   [51]

private static MyFolderRenderer myFolderRenderer;

private MyFolderRenderer() { }

public static FolderRendererHook getInstance()  {              [55]
    // implements the Singleton patten
    if (myFolderRenderer == null)  {
       synchronized (MyFolderRenderer.class) {
          if (myFolderRenderer == null) {
             myFolderRenderer = new MyFolderRenderer();
          }
       }
       return myFolderRenderer;
    }                                                          [57]
}

// implement the invoke specified in the FolderRendererHook interface
public Element invoke(Folder folder, ServiceContext context) {
    // check whether the "MyFolderRendereredContext" has been registered  [60]
    // with the current session
    Session session = context.getSession();
    MyFolderRendererContext renderContext = (MyFolderRendererContext)
                        session.getManagedContext("MyFolderRendererContext");
    if (renderContext == null) {
       synchronized (this) {
           if (renderContext == null) {
              // if not, create the RendererContext and register it
              // with the current seession
              renderContext = new MyFolderRendererContext();
              context.getSession().setManagedContext("MyFolderRendererContext",
                        renderContext);
           }
       }
    }
    // retrieve the new displaying order for the folder's content       [63]
    Service[] newOrder = renderContext.getNewOrder(folder,
                        context.getSession().getUser());
    Document owner = context.getXMLDocument();
    Element simpleResult = PAPrimitive.createSimpleResult(owner, null);
    Element simpleContainer = PAPrimitive.createSimpleContainer(owner,
```

```
                            Services");
        simpleResult.appendChild(simpleContainer);
        if (newOrder != null && newOrder.length >0) {
            // construct the SimpleResult XML to display folder's content
            Element simpleMenu = PAPrimitive.createSimpleMenu(owner, "folder",
                            folder.getName());
            for (int i=0; i < newOrder.length; i++) {
                 String link = newOrder[i].getURLPathParameter();
                 Element menuItem = PAPrimitive.createSimpleMenuItem(owner,
                            newOrder[i].getName(), link, false);
                 simpleMenu.appendChild(menuItem);
            }
            simpleContainer.appendChild(simpleMenu);
        } else {
            // display a message if the folder's content is empty
            Element simpleText = PAPrimitive.createSimpleText(owner);
            Element error = PAPrimitive.createSimpleTextItem(owner,
                            "Error", "No service items found");
            simpleText.appendChild(error);
            simpleContainer.appendChild(simpleText);
        }
        return simpleResult;
    }
    }
```

In line 51, MyFolderRenderer class implements the oracle.panama.rt.hook.FolderRendererHook interface.

Between lines 55 and 57, the class implements the Singleton pattern.

As the class needs to connect to the database to examine the ptg_service_log table, it binds the session-long customized ManagedContext to the Wireless Edition session object in the code between lines 60 and 63.

This class is stateless as it does not have any instance variables. All the state information is stored in the session object as discussed earlier in this document to achieve thread-safe and high-concurrency objectives.

The MyFolderRendererContext.java Source Code:

```
package hook;

import oracle.panama.rt.hook.FolderRendererHook;
```

```
import oracle.panama.rt.ServiceContext;
import oracle.panama.rt.ManagedContext;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import oracle.panama.model.Service;
import oracle.panama.model.ModelServices;
import oracle.panama.model.MetaLocator;
import oracle.panama.model.User;
import oracle.panama.model.Folder;
import oracle.panama.model.Group;
import oracle.panama.PAPrimitive;
import oracle.panama.core.util.Locator;
import oracle.panama.core.admin.L;

public class  MyFolderRendererContext implements ManagedContext {

Connection conn ;

public MyFolderRendererContext() {
    try {
        // lookup the db.connect.string in the panama's System.properties file
        String connectString = Locator.getInstance().getResource()
                                .getSystem().getString("db.connect.string", "");
        // constrct the JDBC connect string, always use the THIN driver for
        // simplicity
        int i = connectString.indexOf('/');
        String user = connectString.substring(0, i);
        int j = connectString.indexOf('@', i+1);
        String password = connectString.substring(i+1, j);
        String dbname = connectString.substring(j+1);
        StringBuffer connStrBuf = new StringBuffer("jdbc:oracle:thin:");

        connStrBuf.append("@");
        connStrBuf.append(dbname);
        // load the Oracle's JDBC driver
        Class.forName("oracle.jdbc.driver.OracleDriver");

        // connect to the database
        conn = DriverManager.getConnection(connStrBuf.toString(),
                                user, password);
```

```
    } catch (Exception e) {
        L.e(e);
        conn = null;
    }

}

public void invalidate() {                    [37]
    try {
        if (conn != null) {
            conn.close();
        } catch (Exception e) {
            L.e(e);
        }
    }
}                                             [40]

public Service[] getNewOrder(Folder folder, User user) {
    Service[] children = null;
    // retrieve all the accessible user services under the user home folder
    children = folder.getAccessibleUserServices(user);
    if (folder == user.getHomeFolder()) {
        // if it is the user's home folder, append group services
        // to the children array
        Group[] groups = user.getGroups();
        children = (children==null) ? new Service[0]:children;
        if (groups != null && groups.length > 0) {
            for (int i = 0; i < groups.length; i++) {
               Service[] tmpServices = groups[i].getAccessibleUserServices(
                                        user);
               // Make sure tmpServices is not null
               tmpServices = (tmpServices == null) ? new Service[0]:tmpServices;
               Service[] tmp = new Service[children.length +
                                        tmpServices.length];
               System.arraycopy(children, 0, tmp, 0, children.length);
              System.arraycopy(tmpServices, 0, tmp, children.length,
                                        tmpServices.length);
               children = tmp;
            }
        }
    }

    if (children == null || children.length == 0) return children;

    // construct a select
```

```
    // select service_id, count(service_id) from ptg_service_log
    // where user_id = ? and service_id in (?,?,..?)
    // group by service_id order by 2 desc
    try {
        StringBuffer select = new StringBuffer("select service_id,
count(service_id) from ptg_service_log where user_id = ");
        select = select.append(user.getId());
        select = select.append(" and service_id in (");
        for (int i =0; i < children.length; i++) {
             if (i !=0) {
                 select = select.append(", ");
             }
             select = select.append(children[i].getId()+ " ");
        }
        select = select.append(") ");
        select=select.append(" group by service_id order by 2 desc");
        Statement st = conn.createStatement();
        L.e("select = "+select.toString());
        ResultSet res = st.executeQuery(select.toString());
        Service[] newOrders = new Service[children.length];
        int j=0;
        ModelServices modelServices= MetaLocator.getInstance()
                                        .getModelServices();
        while (res.next()) {
             newOrders[j++] = modelServices.lookupService(res.getLong(1));
        }
        res.close();
        st.close();
        // append the unvisited services at the end
        for (int k=0; k < children.length; k++) {
             int m=0;
             for (m=0; m <j; m++) {
                 if (newOrders[m].getId() == children[k].getId()) {
                     // already in the new order list
                     break;
                 }
             }
             if (m ==j) {
                 //not in the new order list
                 newOrders[j++] = children[k];
             }
        }
        return newOrders;
   } catch  (Exception e) { L.e(e); return children; }
```

```
    }
}
```

This class encapsulates the folder renderer's context information, particularly the database connection object.

This class implements the `oracle.panama.rt.ManagedContext`. When the session expires, the ManagedContext object is invalidated through the invalidate method between line 37 and line 40. The ManagedContext object is invalidated so that the database connection can be closed.

In addition to the Java source code, you should also modify the following entry in the System.properties file to

```
locator.folder.renderer.hook.class=hook.MyFolderRenderer
```

6.8.1.2 Example 2

The second example is also a hook example, but it takes advantage of the policy concept. The MyAuthenticator first examines the "badguys" table to make sure the login Wireless Edition user is not in the table. If the user is in the table, then the hook rejects the login request. Otherwise, it resumes the default policy implementation in lines 42 and 44.

```
package hook;

import oracle.panama.rt.hook.AuthenticationHook;
import oracle.panama.rt.hook.AuthenticationPolicy;
import oracle.panama.rt.hook.AuthenticationContext;
import oracle.panama.rt.hook.AuthenticationException;
import oracle.panama.rt.hook.AuthenticationFailOverException;
import oracle.panama.rt.Request;
import oracle.panama.rt.hook.AuthenticationContext;
import oracle.panama.core.util.Locator;
import oracle.panama.core.admin.L;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;


public class MyAuthenticator implements AuthenticationHook {
```

```
private static MyAuthenticator myAuthenticator;
private Connection conn;
private PreparedStatement st;

private MyAuthenticator () {
    try {
        // lookup the db.connect.string in the panama's System.properties file
        String connectString =
Locator.getInstance().getResource().getSystem().getString("db.connect.string",
"");
        // constrct the JDBC connect string, always use the THIN driver for
        // simplicity
        int i = connectString.indexOf('/');
        String user = connectString.substring(0, i);
        int j = connectString.indexOf('@', i+1);
        String password = connectString.substring(i+1, j);
        String dbname = connectString.substring(j+1);
        StringBuffer connStrBuf = new StringBuffer("jdbc:oracle:thin:");

        connStrBuf.append("@");
        connStrBuf.append(dbname);
        // load the Oracle's JDBC driver
        Class.forName("oracle.jdbc.driver.OracleDriver");

        // connect to the database
        conn = DriverManager.getConnection(connStrBuf.toString(), user,
password);
        st = conn.prepareStatement("select name from badguys where name = ?");
    } catch (Exception e) {
        L.e(e);
        conn = null;
    }
}

public static AuthenticationHook getInstance() {
    if (myAuthenticator == null) {
        synchronized (MyAuthenticator.class) {
            if (myAuthenticator == null) {
                myAuthenticator = new MyAuthenticator();
            }
        }
    }
    return myAuthenticator;
}
```

```
public AuthenticationContext authenticate(String name, String passwd,
                    Request request) throws AuthenticationException,
                     AuthenticationFailOverException {
      boolean badguy;

      if (conn == null)
          return AuthenticationPolicy.authenticateUser(name, passwd, request);

      try {
          st.setString(1, name);
          ResultSet rs = st.executeQuery();
          badguy = rs.next();
      } catch (Exception e) {
           L.e(e);
           return AuthenticationPolicy.authenticateUser(name,
                                      passwd, request);   [42]
      }

      if (badguy) {
          L.e(name+ " is an intruder!");
          throw new AuthenticationException(name+" is an intruder!");
      } else {
          return AuthenticationPolicy.authenticateUser(name,
                                      passwd, request);    [44]
      }
}

}
```

You should also modify the following entry in the System.properties file in the following manner:

```
locator.authentication.hook.class =hook.MyAuthenticator
```

6.8.2 Event Listener Example

The following partial example (the complete "runable" example is under the WE_HOME\sample\listener directory) illustrates how to implement a RequestListener. This RequestListener simply writes the request related information to a log file.

6.8.2.1 Implementing the RequestListener Interface

The RequestListenerSample source file is as follows:

```
/*
*
$Copyright:
Copyright (c) 1999 Oracle Corporation all rights reserved
$
*/

package listener;

import oracle.panama.rt.Request;
import oracle.panama.rt.Response;
import oracle.panama.rt.Session;
import oracle.panama.rt.AttributeCategory;

import oracle.panama.rt.event.RequestEvent;
import oracle.panama.rt.event.ResponseEvent;
import oracle.panama.rt.event.SessionEvent;
import oracle.panama.rt.event.RequestListener;
import oracle.panama.rt.event.ResponseListener;
import oracle.panama.rt.event.SessionListener;
import oracle.panama.rt.event.AbortServiceException;

/**
* Sample request listener to process request event
*
* @version   $Revision: 1.00 $
* @since     Wireless Edition Release 1.0
*/
public class RequestListenerSample implements RequestListener {       [31]

private final static String BEFORE_REQUEST    = "L__L1";
private final static String REQUEST_BEGIN     = "L__L2";
private final static String SERVICE_BEGIN     = "L__L3";
private final static String SERVICE_END       = "L__L4";
private final static String TRANSFORM_BEGIN   = "L__L5";
private final static String TRANSFORM_END     = "L__L6";
private final static String REQUEST_END       = "L__L7";
private final static String AFTER_REQUEST     = "L__L8";

/**
* The event notification before the start of request             [37]
* @param    an event
```

```
*/                                                                    [39]
public void beforeRequest(RequestEvent event) throws AbortServiceException {
ListenerRegistrationHookSample.println("BEFORE REQUEST -- " + event.toString() +
"---" + event.getTimeStamp());
event.put(BEFORE_REQUEST, new Long(event.getTimeStamp()));
event.getRequest().setAttribute(AttributeCategory.RUNTIME, BEFORE_REQUEST, new
Long(event.getTimeStamp()));

}

/**
* The event notification when request begins
* @param    an event
*/

public void requestBegin(RequestEvent event) throws AbortServiceException {
      ListenerRegistrationHookSample.println("REQUEST BEGIN -- " +
                              event.toString() + "---" + event.getTimeStamp());
      event.put(REQUEST_BEGIN, new Long(event.getTimeStamp()));
      event.getRequest().setAttribute(AttributeCategory.RUNTIME, REQUEST_BEGIN,
                              new Long(event.getTimeStamp()));
}

/**
* The event notification when service begins
* @param    an event
*/

public void serviceBegin(RequestEvent event) throws AbortServiceException {
      ListenerRegistrationHookSample.println("SERVICE BEGIN -- " +
                              event.toString() + "---" + event.getTimeStamp());
      event.put(SERVICE_BEGIN, new Long(event.getTimeStamp()));
      event.getRequest().setAttribute(AttributeCategory.RUNTIME, SERVICE_BEGIN,
                              new Long(event.getTimeStamp()));
}

/**
* The event notification when service end
* @param    an event
*/
public void serviceEnd(RequestEvent event) throws AbortServiceException {
      ListenerRegistrationHookSample.println("SERVICE END -- " +
                              event.toString() + "---" + event.getTimeStamp());
      event.put(SERVICE_END, new Long(event.getTimeStamp()));          [62]
      event.getRequest().setAttribute(AttributeCategory.RUNTIME, SERVICE_END,
```

```
                        new Long(event.getTimeStamp()));
}

/**
* The event notification when transform begins                    [65]
* @param    an event
*/

public void transformBegin(RequestEvent event) throws AbortServiceException {
        ListenerRegistrationHookSample.println("TRANSFORM BEGIN -- " +
                            event.toString() + "---" + event.getTimeStamp());
        event.put(TRANSFORM_BEGIN, new Long(event.getTimeStamp()));
        event.getRequest().setAttribute(AttributeCategory.RUNTIME,
                            TRANSFORM_BEGIN, new Long(event.getTimeStamp()));
}

/**
* The event notification when transform end
* @param    an event
*/
public void transformEnd(RequestEvent event) throws AbortServiceException {
        ListenerRegistrationHookSample.println("TRANSFORM END -- " +
                            event.toString() + "---" + event.getTimeStamp());
        event.put(TRANSFORM_END, new Long(event.getTimeStamp()));
        event.getRequest().setAttribute(AttributeCategory.RUNTIME,
                            TRANSFORM_END, new Long(event.getTimeStamp()));
}

/**
* The event notification when request ends
* @param    an event
*/
public void requestEnd(RequestEvent event) throws AbortServiceException {
        ListenerRegistrationHookSample.println("REQUEST END -- " +
                            event.toString() + "---" + event.getTimeStamp());
        event.put(REQUEST_END, new Long(event.getTimeStamp()));
        event.getRequest().setAttribute(AttributeCategory.RUNTIME,
                            REQUEST_END, new Long(event.getTimeStamp()));
}

/**
* The event notification when request error happens
* @param    an event
*/
public void requestError(RequestEvent event) throws AbortServiceException {
```

```
        ListenerRegistrationHookSample.println("REQUEST ERROR -- " +
                                event.toString() + "---" + event.getTimeStamp());
    }

    /**
    * The event notification after the end of request
    * @param    an event
    */
    public void afterRequest(RequestEvent event) throws AbortServiceException {
        ListenerRegistrationHookSample.println("AFTER REQUEST -- " +
                                event.toString() + "---" + event.getTimeStamp());
        event.put(AFTER_REQUEST, new Long(event.getTimeStamp()));
        event.getRequest().setAttribute(AttributeCategory.RUNTIME,
                                AFTER_REQUEST, new Long(event.getTimeStamp()));

        // start logging the object cached in the Request
        ListenerRegistrationHookSample.println("logging the object cached in the
    request");

        Long beforeRequestTime = (Long) event.getRequest().getAttribute(
                                AttributeCategory.RUNTIME, BEFORE_REQUEST);
        if (beforeRequestTime != null)
             ListenerRegistrationHookSample.println("BEFORE REQUEST: " +
                                beforeRequestTime.longValue());

        Long requestBeginTime = (Long) event.getRequest().getAttribute(
                                AttributeCategory.RUNTIME, REQUEST_BEGIN);

        if (requestBeginTime != null)
             ListenerRegistrationHookSample.println("REQUEST BEGIN: " +
                                requestBeginTime.longValue());

        Long serviceBeginTime = (Long) event.getRequest().getAttribute(
                                AttributeCategory.RUNTIME, SERVICE_BEGIN);

        if (serviceBeginTime != null)
              ListenerRegistrationHookSample.println("SERVICE BEGIN: " +
                                serviceBeginTime.longValue());

        Long serviceEndTime = (Long) event.getRequest().getAttribute(
                                AttributeCategory.RUNTIME, SERVICE_END);

        if (serviceEndTime != null)
               ListenerRegistrationHookSample.println("SERVICE END: " +
                                serviceEndTime.longValue());
```

```
        Long transformBeginTime = (Long) event.getRequest().getAttribute(
                                   AttributeCategory.RUNTIME, TRANSFORM_BEGIN);

        if (transformBeginTime != null)
             ListenerRegistrationHookSample.println("TRANSFORM BEGIN: " +
                                   transformBeginTime.longValue());

        Long transformEndTime = (Long) event.getRequest().getAttribute(
                                   AttributeCategory.RUNTIME, TRANSFORM_END);

        if (transformEndTime != null)
             ListenerRegistrationHookSample.println("TRANSFORM END: " +
                                  transformEndTime.longValue());

        Long requestEndTime = (Long) event.getRequest().getAttribute(
             AttributeCategory.RUNTIME, REQUEST_END);

        if (requestEndTime != null)
             ListenerRegistrationHookSample.println("REQUEST END: " +
                                  requestEndTime.longValue());

        Long afterRequestTime = (Long) event.getRequest().getAttribute(
                                  AttributeCategory.RUNTIME, AFTER_REQUEST);

        if (afterRequestTime != null)
             ListenerRegistrationHookSample.println("AFTER REQUEST: " +
                                  afterRequestTime.longValue());

        if ((afterRequestTime != null) && (beforeRequestTime != null))
             ListenerRegistrationHookSample.println("REQUEST DURATION: " +
                                  (afterRequestTime.longValue() -
                                    beforeRequestTime.longValue()));

        // start logging the object cached in the RequestEvent
        ListenerRegistrationHookSample.println("logging the object cached in the
request event");

        beforeRequestTime = (Long) event.get(BEFORE_REQUEST);
        if (beforeRequestTime != null)
              ListenerRegistrationHookSample.println("BEFORE REQUEST EVENT: " +
                                    beforeRequestTime.longValue());
```

```
        requestBeginTime = (Long) event.get(REQUEST_BEGIN);
        if (requestBeginTime != null)
             ListenerRegistrationHookSample.println("REQUEST BEGIN EVENT: " +
                                        requestBeginTime.longValue());

        serviceBeginTime = (Long) event.get(SERVICE_BEGIN);
        if (serviceBeginTime != null)
             ListenerRegistrationHookSample.println("SERVICE BEGIN EVENT: " +
                                        serviceBeginTime.longValue());

        serviceEndTime = (Long) event.get(SERVICE_END);
        if (serviceEndTime != null)
             ListenerRegistrationHookSample.println("SERVICE END EVENT: " +
                                        serviceEndTime.longValue());

        transformBeginTime = (Long) event.get(TRANSFORM_BEGIN);
        if (transformBeginTime != null)
             ListenerRegistrationHookSample.println("TRANSFORM BEGIN EVENT: " +
                                        transformBeginTime.longValue());

        transformEndTime = (Long) event.get(TRANSFORM_END);
        if (transformEndTime != null)
             ListenerRegistrationHookSample.println("TRANSFORM END EVENT: " +
                                        transformEndTime.longValue());

        requestEndTime = (Long) event.get(REQUEST_END);
        if (requestEndTime != null)
             ListenerRegistrationHookSample.println("REQUEST END EVENT: " +
                                        requestEndTime.longValue());

        afterRequestTime = (Long) event.get(AFTER_REQUEST);
        if (afterRequestTime != null)
             ListenerRegistrationHookSample.println("AFTER REQUEST EVENT: " +
                                        afterRequestTime.longValue());

        if ((afterRequestTime != null) && (beforeRequestTime != null))
             ListenerRegistrationHookSample.println("REQUEST DURATION EVENT: " +
             (afterRequestTime.longValue() - beforeRequestTime.longValue()));

    }

    }
```

Line 31 in the above code example declares the implementation of the `oracle.panama.rt.event.RequestListener` interface.

6.8.2.2 Register the Request Listener Entry in the System.properties File

Append RequestListenerSample to locator.request.listener.classes entries as follows:

```
locator.request.listener.classes = …,…, RequestListenerSample
```

You can register multiple listener classes for each event category. Each listener class name is separated by a comma.

6.8.2.3 Register the RequestListener with Each Request Object

You should implement the ListenerRegistrationHook to register your request listener object whenever a new request is created. See the code section between line 62 and line 65 in the code example below.

Your new registration hook class has to implement the oracle.panama.rt.event.ListenerRegistrationHook interface as in line 31 in the code example below. The class also needs to implement the Singleton pattern. See the code section between lines 37 and 39 in the code example below.

```
/*
 * $Header:
/home/cvsroot/panama/src/core/oracle/panama/rt/common/ListenerRegistration.java
 *
 * $Copyright:
 *             Copyright (c) 1999 Oracle Corporation all rights reserved
 * $
 */

package listener;

import java.io.FileOutputStream;
import java.io.PrintStream;
import java.io.FileNotFoundException;
import java.net.URL;

import oracle.panama.rt.Request;
import oracle.panama.rt.Response;
import oracle.panama.rt.Session;

import oracle.panama.rt.event.RequestListener;
import oracle.panama.rt.event.ResponseListener;
import oracle.panama.rt.event.SessionListener;

import oracle.panama.rt.hook.ListenerRegistrationHook;
```

```
import oracle.panama.rt.hook.ListenerRegistrationPolicy;

/**
Sample listener hook implementation to register/unregister SessionListener,
RequestListner, ResponseListener
Default event listeners, such as SystemLogger, can be registered through
ListenerRegistrationPolicy

*
to try this sample, you need to add the following line in jserv.properties
*
wrapper.classpath=i:\panama\pub\examples\sample
*
* @version   $Revision: 1.00 $
* @since     Wireless Edition Release 1.0
*/
public final class ListenerRegistrationHookSample implements      [31]
ListenerRegistrationHook {

public final static String LISTENER_LOG_FILE = "ListenerSample.log";
public static PrintStream logPrint = System.out;

private SessionListener sessionListener = null;
private RequestListener requestListener = null;
private ResponseListener responseListener = null;

private static ListenerRegistrationHookSample singleInstance = null;

public static ListenerRegistrationHookSample getInstance() {       [37]
     if (singleInstance == null) {
          singleInstance = new ListenerRegistrationHookSample();
     }
     return singleInstance;
}                                                                  [39]

public void finalize() {
     logPrint.println("RegistrationHook is deallocated -- " +
                                 System.currentTimeMillis());
     logPrint.flush();
     logPrint.close();
}

public static void println(String str) {
     logPrint.println(str);
     logPrint.flush();
```

```
}

private ListenerRegistrationHookSample() {
     URL url = ClassLoader.getSystemResource(
                              "listener/ListenerRegistrationHookSample.class");
     if (url != null) {
          String filePath = url.getFile();
          int lastSlash = filePath.lastIndexOf("/");
          filePath = filePath.substring(1, lastSlash);

          filePath = filePath + "/" + LISTENER_LOG_FILE;
          try {
              FileOutputStream logFile = new FileOutputStream(filePath, true);
              logPrint = new PrintStream(logFile);
          } catch (Exception fnfe) {
              fnfe.printStackTrace();
          }
      }
      logPrint.println("RegistrationHook is initialized -- " +
          System.currentTimeMillis());
      logPrint.flush();
}

/**
* instantiate the sample session listener class and register to sesson
* @param request    an incoming request
* @param session    a new session to register listeners
*/
public void registerSessionListeners(Request request, Session session) {
      sessionListener = new SessionListenerSample();
      if (sessionListener != null) {
          session.addSessionListener(sessionListener);
      }

     // optional, register default session listeners
     ListenerRegistrationPolicy.registerSessionListeners(request, session);
}

/**
* instantiate the sample request listener class and register to request
* @param request    a new request to register listeners
*/
public void registerRequestListeners(Request request) {          [62]
      requestListener = new RequestListenerSample();
      if (requestListener != null) {
```

```
            request.addRequestListener(requestListener);
        }
        //optional, register default request listeners
        ListenerRegistrationPolicy.registerRequestListeners(request);
}

/**                                                                    [65]
* instantiate the sample response listener class and register to response
* @param request     an incoming request
* @param session     an existing session
* @param response    a new response to register listeners
*/
public void registerResponseListeners(Request request, Response response) {
        responseListener = new ResponseListenerSample();
        if (responseListener != null) {
            response.addResponseListener(responseListener);
        }
        // optional, register default response listeners
        ListenerRegistrationPolicy.registerResponseListeners(request, response);
}

/**
* unregister the listeners from session.
* @param session     a session to unregister listeners
*/
public void unregisterSessionListeners(Session session) {
        if (sessionListener != null) {
            session.removeSessionListener(sessionListener);
        }
        //optional, unregister default session listeners
        ListenerRegistrationPolicy.unregisterSessionListeners(session);
}

/**
* unregister the listeners from request.
* @param request     a request to unregister listeners
*/
public void unregisterRequestListeners(Request request) {
        if (requestListener != null) {
             request.removeRequestListener(requestListener);
        }
        //optional, unregister default request listeners
        ListenerRegistrationPolicy.unregisterRequestListeners(request);
}
```

```
/**
* unregister the listeners from response.
* @param response    a response to unregister listeners
*/
public void unregisterResponseListeners(Response response) {
     if (responseListener != null) {
          response.removeResponseListener(responseListener);
     }
     //optional, unregister default response listeners
     ListenerRegistrationPolicy.unregisterResponseListeners(response);
}

}
```

6.8.2.4 Modify the Event Mask

Since the sample request is interested in all the request events, you should make sure that the event mask for all the request-related events is set to true in the System.properties file as follows:

```
event.before.request=true
event.request.begin=true
event.request.end=true
event.service.begin=true
event.service.end=true
event.transform.begin=true
event.transform.end=true
event.request.error=true
event.after.request=true
```

7

Using the Data Model API

This document describes the Data Model API. Each section of this document presents a different topic. These sections include:

- Section 7.1, "Overview"
- Section 7.2, "Class Hierarchy"
- Section 7.3, "Sample Adapter that Uses the Data Model API"

7.1 Overview

To support interaction with persistent objects programmatically, the Wireless Edition Data Model API provides a set of Java interfaces that constitute a high-level interface to the underlying Wireless Edition Data Model. By implementing the Java interfaces in the Data Model API, you can create, delete, modify, and query Wireless Edition persistent objects.

The intended users of the public interfaces in the Data Model API are developers of services, adapters, and transformers. Developers can implement the interfaces in the Data Model API to develop stand-alone applications which manipulate persistent objects. Developers can also implement the interfaces from their customized Java adapters or Java Server Pages (JSP).

Although these interfaces will preserve the data integrity for the trusted developers, these interfaces will not enforce any security on who can do what. There is no enforcement of authentication and authorization policies. Developers should apply extreme caution when developing services using the interfaces in the Data Model API, and should take appropriate measures to prevent any undesired side effects when these services are invoked by the end users.

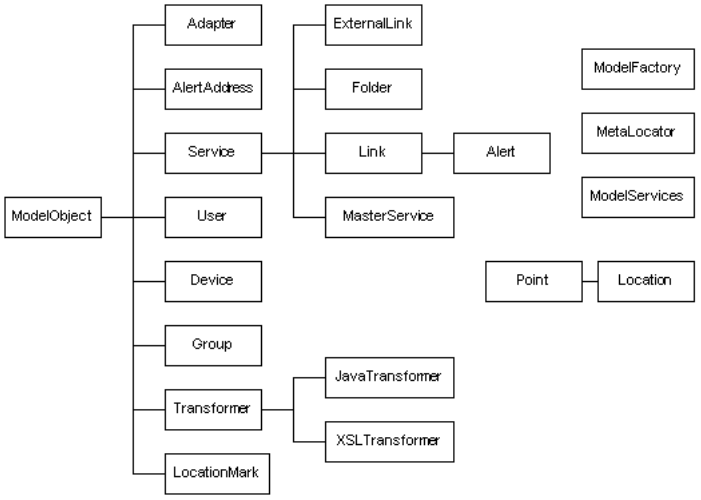
7.2 Class Hierarchy

The following sections describe the interfaces within the class hierarchy in the Data Model API. These interfaces are contained in the `oracle.panama.model` package. For a sample application that implements some of the interfaces, see Section 7.3, "Sample Adapter that Uses the Data Model API".

7.2.1 ModelObject

The ModelObject is the root interface for managing all of the "generic" aspects of a persistent object. It resides in the `oracle.panama.model` package. The figure below illustrates the structural relationships among all of the interfaces within the `oracle.panama.model` package.

Figure 7–1 The Data Model API Class Hierarchy



The subinterfaces in the ModelObject class hierarchy are all persistent objects. These subinterfaces are (in alphabetical order):

- Adapter
- Alert
- AlertAddress
- Device

- ExternalLink
- Folder
- Group
- JavaTransformer
- Link
- LocationMark
- MasterService
- Service
- Transformer
- User
- XSLTransformer

The following sections describe each subinterface.

7.2.1.1 Adapter

Adapter extends the ModelObject root interface. Adapter is the container for the RuntimeAdapter. It is the interface that is to be implemented by all adapters. It inherits methods from the interface `oracle.panama.model.ModelObject`.

7.2.1.2 AlertAddress

AlertAddress extends the ModelObject root interface. An AlertAddress is the device-specific address, such as a phone or an email address.

For example, an AlertAddress can be a phone number of the logical device SMS or a the same phone number of the logical device WAP. The AlertAddress information is used for sending asynchronous Alert results to an address, or to authenticate a user from an incoming request.

AlertAddress inherits methods from the interface `oracle.panama.model.ModelObject`.

7.2.1.3 Device

Device extends the ModelObject root interface. A Device is the definition of the target logical device protocol. It can, for example, be WML11 for WML specific

devices, but also WML_Nokia7110 for more Nokia specific WML. Other examples are SMS and EMAIL.

Device inherits methods from the interface `oracle.panama.model.ModelObject`.

Observe that the same physical device can support multiple logical devices; a phone, for example, can support both the SMS and the WAP protocols.

7.2.1.4 Group

Group extends the ModelObject root interface. A Group is a collection of users. It is used to publish specific services to the group members. A user can access those services that are accessible to the group to which the user belongs.

Group inherits methods from the interface `oracle.panama.model.ModelObject`.

7.2.1.5 LocationMark

LocationMark, which is in the `oracle.panama.model` package, extends the ModelObject root interface. It also extends the Location class. It is a persistent object that represents the named and geocoded correct physical address.

LocationMark inherits methods from the interface `oracle.panama.model.ModelObject`.

7.2.1.6 MetaLocator

MetaLocator, which is in the `oracle.panama.model` package, extends the `java.lang.Object` interface. The MetaLocator is used to look up factories.

7.2.1.7 ModelFactory

ModelFactory, which is in the `oracle.panama.model` package, provides the entry point to create model objects.

7.2.1.8 ModelServices

ModelServices, which is in the `oracle.panama.model` package, provides the entry point to find model objects.

7.2.1.9 Service

Service extends the ModelObject root interface. Service is an "abstract" interface and handles all generic aspects of a service.

It inherits methods from the interface `oracle.panama.model.ModelObject`.

It contains the following subinterfaces:

- ExternalLink — ExternalLink extends Service. An ExternalLink is a reference to an external URL.
- Folder — Folder extends the Service interface. A Folder is like a directory in a file system; it contains other services including other sub-folders. The model for a Folder is one that follows the "standardized" composite pattern, where the Folder is an aggregate of other services.
- Link — Link extends the Service interface. A Link is a pointer to any other service "including" another Link. The Link is used to "customize" master services or to create private tree structures of accessible master services. It can override any accessible parameter kept by the service "chain" down to the final master service. Link contains the subinterface Alert.
  - Alert — Alert extends the Link interface. An Alert (sometimes referred to as a Job) is a service which is set to be automatically executed, given a particular time interval specification. The Alert interface inherits methods from the following interfaces
    - `oracle.panama.model.Link`
    - `oracle.panama.model.Service`
    - `oracle.panama.model.ModelObject`
- MasterService — MasterService extends the Service interface. The MasterService is the "final" Service. It is the template for all other Services. It always uses an Adapter to communicate with the external source.

7.2.1.10 Transformer

Transformer extends the ModelObject root interface. Transformer is the abstract base interface for all transformation sub-classes. It acts as a bridge between the real transformation implementation (Java or XSL) and the clients.

It inherits methods from the interface `oracle.panama.model.ModelObject`.

It has the following subinterfaces:

- JavaTransformer — JavaTransformer extends the Transformer interface. A JavaTransformer is a class that implements the Transformer interface and is expected to handle the transformation from the SimpleResult DTD to the device-specific markup language.

- XSLTransformer — XSLTransformer extends the Transformer interface. An XSLTransformer is a stylesheet which is expected to handle the transformation from the SimpleResult DTD to the device-specific markup language.

7.2.1.11 User

The User interface extends the root interface ModelObject. The User interface represents the identity of the current actual user.

Each actual user has its own home folder. The actual user can access those services that are accessible to the group to which the user belongs.

The implementation of the User interface may access another provisioning system to manage the information about the current actual user.

It inherits methods from the interface `oracle.panama.model.ModelObject`.

7.3 Sample Adapter that Uses the Data Model API

The following sample adapter illustrates how you can create new objects by implementing the interfaces in the Data Model API and by using the methods associated with those interfaces. The purpose of the sample is not to show how to create an adapter, but rather how to create a new object (in this case, a new user).

The main focus in this sample adapter is on demonstrating the usage of the methods to create, search for, and finally delete an object.

The complete listing of the sample code is in Section 7.3.1, "Sample Adapter Code". The numbers that appear in brackets next to a line of code in the listing are referenced in the discussion to correlate the explanation with the corresponding lines in the code itself.

- To create a new object (such as a new user), you use ModelFactory.
- To search for an object, you use ModelServices.
- To get started, you reference the implementation of ModelFactory and ModelServices (reference **[1]**).

This is how you do it:

```
MetaLocator metaLocator = MetaLocator.getInstance();
modelFactory = metaLocator.getModelFactory();
modelServices = metaLocator.getModelServices();
```

The MetaLocator interface is used to lookup the ModelFactory and ModelServices. The `getInstance()` method in this interface gets the single instance of this

MetaLocator. The method returns the single instance of the MetaLocator. The methods `getModelFactory` and `getModelServices` look up the ModelFactory and the ModelServices.

Typically, to create a new object, you should check first if the object already exists. To look up any object, you use the ModelServices interface and the method `lookupX(java.lang.String name)`, where `X` is the class name of the object. In this sample adapter, to create a new user (the code section for creating a new user starts in reference **[2]**), you first look up the user by using the `lookupUser(userName)` method in the ModelServices interface (reference **[3]**), as the following line of code shows:

```
modelServices.lookupUser(userName);
```

This is the first step in creating any new persistent object in the Data Model.

The `lookupUser(userName)` method searches for the user by name and, if the user name is found, returns the name of the user. If the user name cannot be found, the method throws a `PanamaRuntimeException`.

Next, you check if the group to which the user belongs (or should belong) already exists (reference **[4]**). Following the procedure for looking up any object, you use the ModelServices interface and the `lookupGroup(groupName)` method to look up a group by name. If the group is found, the method returns the name of the group. If the group is not found, the method throws a `PanamaRuntimeException`.

After checking if the user and the group already exist, you create the new user object (reference **[5]** to reference **[6]**):

```
     user = modelFactory.createUser(userName, groups);
} else {
     user = modelFactory.createUser(userName);
}
user.setPassword(userPassword);
user.setEnabled(true);
```

You must save the newly created user. Each newly created object must be saved after it is created (reference **[7]**):

```
modelFactory.save();
```

This saves your work. Save applies to all created or modified objects in the current thread. The objects are saved to the persistent storage and the transaction is committed. The method throws `PanamaException` when it is unable to save the work.

To search for an object after creating it, use the `searchElement` method with the criteria specified in the form of the parameters `rootElement, tagname, atrr1, value1`. The method returns `Element`. In the sample, to search for the user object, you use the following method (reference **[8]**):

```
private void searchUser(Document owner, Element container, Arguments
args)
```

To get a set of user names (for example, all the names that start with the letter "B"), you work with enumeration. Use `ResultSetEnumeration` (reference **[9]**) and the method `findUsers` (reference **[10]**). The method `findUsers` pluralizes users because it is using a template. This is used for getting a set of names; it will return several names. See also references **[11]** and **[12]** in the listing of the complete sample code.

You should close the `ResultSetEnumeration` (reference **[13]**). It is associated with the database cursor. If you do not close it, you leave a database cursor open.

To delete an object, you use the `deleteElement` method and the code that matches the code section in reference **[14]**. In this case, to delete the user you use `deleteUser` and the specified criteria as the parameters. Note references **[15]** and **[16]**. The user name must be exact, because if it is a template, the method will throw an exception.

7.3.1 Sample Adapter Code

```
/*
 * $Copyright:
 *             Copyright (c) 2001 Oracle Corporation all rights reserved
 * $
 */

package model;

import java.util.Vector;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import oracle.panama.Arguments;
import oracle.panama.ArgumentType;
import oracle.panama.InputArgument;
import oracle.panama.OutputArguments;
import oracle.panama.PanamaRuntimeException;
import oracle.panama.PAPrimitive;
import oracle.panama.adapter.AdapterException;
import oracle.panama.adapter.RuntimeAdapter;
```

```
import oracle.panama.adapter.RuntimeAdapterHelper;
import oracle.panama.model.Group;
import oracle.panama.model.MetaLocator;
import oracle.panama.model.ModelFactory;
import oracle.panama.model.ModelServices;
import oracle.panama.model.ResultSetEnumeration;
import oracle.panama.model.User;
import oracle.panama.rt.Request;
import oracle.panama.rt.ServiceContext;

/**
 * This is a sample adapter demonstrating the usage of the model API.
 *
 * @since     Oracle9i Aplication Server Wireless Edition
 * @version   1.0
 */
public class SampleModelAdapter implements RuntimeAdapter {

    /** Command variable */
    private static final String COMMAND     = "PAsection";
    /** Commands */
    /** Add new user */
    private static final String CREATE_USER = "CreateUser";
    /** Remove a user */
    private static final String DELETE_USER = "DeleteUser";
    /** Search for a user */
    private static final String SEARCH_USER = "SearchUser";
    /** The list of all top level commands */
    private static final String[] COMMANDS  =
        new String[] {CREATE_USER, SEARCH_USER};

    private static final String USER_NAME         = "Name";
    private static final String USER_GROUP        = "Group";
    private static final String USER_PASSWORD     = "Password";
    private static final String[] CREATE_ARGUMENTS  = new String[] {
        USER_NAME, USER_GROUP, USER_PASSWORD};
    private static final String[] SEARCH_ARGUMENTS = new String[] {
        USER_NAME};

    private boolean initialized;
    private Arguments initArguments;
    private Arguments inputArguments;
    private OutputArguments outputArguments;

    private ModelFactory modelFactory;
```

```
private ModelServices modelServices;

public SampleModelAdapter() {
    initialized = false;
    MetaLocator metaLocator = MetaLocator.getInstance();      [1]
    modelFactory = metaLocator.getModelFactory();
    modelServices = metaLocator.getModelServices();
}

/**
 * Get the init arguments
 * @return  init arguments
 */
public Arguments getInitArguments() throws AdapterException {
    if (initArguments == null) {
        synchronized (this) {
            if (initArguments == null) {
                initArguments = RuntimeAdapterHelper.createArguments();
            }
        }
    }
    return initArguments;
}

/**
 * Get the input Arguments
 * @return    input arguments
 */
public Arguments getInputArguments() throws AdapterException {
    checkState();
    return inputArguments;
}

/**
 * Get the output Arguments
 * @return    an array of output arguments
 */
public OutputArguments getOutputArguments() throws AdapterException {
    checkState();
    return outputArguments;
}

/**
 * Initialize the adapter using the information from the init arguments.
 * @param args    init arguments
```

```
     */
    public void init(Arguments args) throws AdapterException {
        if (initialized == false) {
            synchronized (this) {
                if (initialized == false) {
                    initialized = true;
                    // Create Input Arguments
                    inputArguments = RuntimeAdapterHelper.createArguments();

                    InputArgument arg = inputArguments.createInput(COMMAND);
                    arg.setComment("Command to Adapter. Either 'CreateUser' or


'SearchUser'");
                    arg.setType(ArgumentType.ENUM);
                    arg.setOptions(COMMANDS);

                    arg = inputArguments.createInput(USER_NAME);
                    arg.setComment("User Name");
                    arg.setType(ArgumentType.SINGLE_LINE);

                    arg = inputArguments.createInput(USER_GROUP);
                    arg.setComment("User Group");
                    arg.setType(ArgumentType.ENUM);
                    arg.setOptions(getGroupNames());

                    arg = inputArguments.createInput(USER_PASSWORD);
                    arg.setComment("User Password");
                    arg.setType(ArgumentType.SINGLE_LINE);

                    // Create Output Arguments
                    outputArguments =
RuntimeAdapterHelper.createOutputArguments();
                }
            }
        }
    }

    /**
     * Invoke the adapter using the input and output parameters in the
     * service context.
     * @param context   the context that contains input parameters
     */
    public Element invoke(ServiceContext context) throws AdapterException {
        checkState();
```

```
            Document doc = context.getXMLDocument();
            Element simpleResult = PAPrimitive.createSimpleResult(doc, null);
            Element simpleContainer = PAPrimitive.createSimpleContainer(doc, null);
            simpleResult.appendChild(simpleContainer);

            Arguments args = context.getInputArguments();
            String command = args.getInputValue(COMMAND);
            if (command == null || command.length() == 0) {
                // Create the main menu
                createMenu(doc, simpleContainer, args);
            } else if (command.equals(CREATE_USER)) {
                // Create a new user
                createUser(context, doc, simpleContainer, args);
            } else if (command.equals(SEARCH_USER)) {
                searchUser(doc, simpleContainer, args);
            } else if (command.equals(DELETE_USER)) {
                deleteUser(doc, simpleContainer, args);
            } else {
                Element errMsg = simpleText(doc, "Unknown command", "Error");
            }
            return simpleResult;
        }

        /**
         * Destroy the provider.
         */
        public void destroy() {}

    //
    // Helper methods
    //
        /**
         * Check if the state of this Adapter is all right.
         * @exception AdapterException when the state is illegal
         */
        private synchronized void checkState() throws AdapterException {
            if (!initialized) {
                throw new AdapterException("SampleModelAdapter is not initialized");
            }
        }

        /**
         * Find all group names
         * @return String[] with names
         */
```

```
private String[] getGroupNames() throws AdapterException {
    String[] names;
    ResultSetEnumeration result = null;
    try {
        // Find all user groups - use a wildcard for the name expression
        result = modelServices.findGroups("*");
        Vector buffer = new Vector();
        while (result.hasMoreElements()) {
            Group group = (Group)result.next();
            String name = group.getName();
            buffer.addElement(name);
        }
        names = new String[buffer.size()];
        buffer.copyInto(names);
    } catch (PanamaRuntimeException ex) {
        throw new AdapterException(ex);
    } finally {
        if (result != null) {
            result.close();
            result = null;
        }
    }
    return names;
}

/**
 * Create a simple menu. <p>
 *
 * There are two choices. Create user or search for users.
 *
 * @param owner the owner document.
 * @param container   an output parameter for the menu.
 * @param args  the arguments.
 */
private void createMenu(Document owner, Element container, Arguments args) {
    Element menu = PAPrimitive.createSimpleMenu(owner, "");
    container.appendChild(menu);
    String target = RuntimeAdapterHelper.getURLPAoidParameter(args);
    target = target + ((target.indexOf('?') == -1) ? "?" : "&");
    //create the menuitem with submenu
    for (int i = 0; i < COMMANDS.length; i++) {
        Element menuItem = PAPrimitive.createSimpleMenuItem(
            owner, COMMANDS[i],
            target + Request.SECTION + "=" + COMMANDS[i], false);
        menu.appendChild(menuItem);
```

```
        }
    }

    /**                                          [2]
     * Create a new user. <p>
     *
     * The service request contains the parameters.
     *
     * @param context the ServiceContext
     * @param owner the owner document.
     * @param container    an element for the output result.
     * @param args    the service request holds the arguments for the user.
     */
    private void createUser(ServiceContext context, Document owner, Element
container,


Arguments args) {
        try {
            String[] missingArgs = getMissingCreateArguments(args);
            if (missingArgs == null || missingArgs.length == 0) {
                // There are no missing arguments - create the new user
                String userName = args.getInputValue(USER_NAME);
                String groupName = args.getInputValue(USER_GROUP);
                String userPassword = args.getInputValue(USER_PASSWORD);
                // First check if the user does not already exists
                try {
                    modelServices.lookupUser(userName);                  [3]
                    // If we are here the user must already exists
                  Element errTxt = simpleText(owner, "The user already
exists", "Error");
                    container.appendChild(errTxt);
                    return;
                } catch (PanamaRuntimeException ignore) {}

                Group group = null;
                try {
                    // A PanamaRuntimeException will be thrown if the group is
                    // not found
                    group = modelServices.lookupGroup(groupName);         [4]
                } catch (PanamaRuntimeException ex) {
                    group = null;
                }
                User user;
                // NOTE: modelFactory.createUser() will automatically create a
```

```
                // home folder for the new user.
                if (group != null) {
                    Group[] groups = new Group[1];
                    groups[0] = group;
                    user = modelFactory.createUser(userName, groups);   [5]
                } else {
                    user = modelFactory.createUser(userName);
                }
                user.setPassword(userPassword);
                user.setEnabled(true);                                   [6]

                // Do not forget to save the newly created object
                modelFactory.save();                                     [7]

                Element successTxt = simpleText(owner, "User \"" + userName +
"\" was


created!", "Success");
                container.appendChild(successTxt);
            } else {
                //construct the target parameters
                String target = RuntimeAdapterHelper.getURLPAoidParameter(args);
                target = target.concat("&" + COMMAND + "=" + CREATE_USER);
                Element form = PAPrimitive.createSimpleForm(owner,
                    args.getInputValue(ServiceContext.INP_FIRST_SERVICE_NAME),
target);
                container.appendChild(form);
                RuntimeAdapterHelper.createInputFields(form, missingArgs, args);
            }
        } catch (Exception ex) {
            Element errTxt = simpleText(owner, ex.getMessage(), "Error");
            container.appendChild(errTxt);
        }
    }

    /**
     * Check that arg has an value for all valid attributes.
     *
     * @param arg Arguments which holds the parameters
     * @exception Exception when required input parameters are missing
     */
    private String[] getMissingCreateArguments(Arguments args) {
        String[] missingArgs;
        Vector buffer = new Vector();
```

```
        for (int i = 0; i < CREATE_ARGUMENTS.length; i++) {
            String argValue = args.getInputValue(CREATE_ARGUMENTS[i]);
            if (argValue == null || argValue.length() == 0) {
                buffer.addElement(CREATE_ARGUMENTS[i]);
            }
        }
        missingArgs = new String[buffer.size()];
        buffer.copyInto(missingArgs);
        return missingArgs;
    }

    /**
     * Search for users. <p>
     *
     * The service request contains the parameters.
     *
     * @param owner the owner document.
     * @param container    an element for the output result.
     * @param args    the service request holds the arguments for the user.
     */
    private void searchUser(Document owner, Element container, Arguments
args) {                                                                    [8]
        try {
            String[] missingArgs = getMissingSearchArguments(args);
            if (missingArgs == null || missingArgs.length == 0) {
                // There are no missing arguments - search for the user
                String userName = args.getInputValue(USER_NAME);

                // Create the table
                Element table = owner.createElement(PAPrimitive.TAG_
SIMPLETABLE);
                container.appendChild(table);
                Element tableHeader =


owner.createElement(PAPrimitive.TAG_SIMPLETABLEHEADER);
                table.appendChild(tableHeader);
                Element headerCol = owner.createElement(PAPrimitive.TAG_
SIMPLECOL);
                headerCol.appendChild(owner.createTextNode("Name"));
                tableHeader.appendChild(headerCol);
                headerCol = owner.createElement(PAPrimitive.TAG_SIMPLECOL);
                headerCol.appendChild(owner.createTextNode("Delete?"));
                tableHeader.appendChild(headerCol);
```

```
                String target = RuntimeAdapterHelper.getURLPAoidParameter(args);
                target = target + ((target.indexOf('?') == -1) ? "?" : "&");

                ResultSetEnumeration result = null;                    [9]
                try {
                    result = modelServices.findUsers(userName);        [10]
                    // NOTE: A real adapter should not fetch all users at once.
                    // Instead it should provide a "More..." button.
                    // But this is not the goal of this example ;-)
                    int count = 0;
                    while (result.hasMoreElements() && count < 20) {   [11]
                        count++;
                        User user = (User) result.next();              [12]
                        Element row = owner.createElement(PAPrimitive.TAG_
SIMPLEROW);
                        table.appendChild(row);
                        Element col = owner.createElement(PAPrimitive.TAG_
SIMPLECOL);
                        row.appendChild(col);
                        String name = user.getName();
                        col.appendChild(owner.createTextNode(name));

                        col = owner.createElement(PAPrimitive.TAG_SIMPLECOL);
                        row.appendChild(col);
                        Element delete = owner.createElement(PAPrimitive.TAG_
SIMPLEHREF);
                        col.appendChild(delete);
                        delete.appendChild(owner.createTextNode("delete"));
                        delete.setAttribute(PAPrimitive.ATTR_TARGET, target +


Request.SECTION +
                            "=" + DELETE_USER + "&" + USER_NAME + "=" + name);
                    }
                } catch (PanamaRuntimeException ex) {
                } finally {
                    if (result != null) {
                        result.close();                        [13]
                        result = null;
                    }
                }
            } else {
                //construct the target parameters
                String target = RuntimeAdapterHelper.getURLPAoidParameter(args);
                target = target.concat("&" + COMMAND + "=" + SEARCH_USER);
```

```
                Element form = PAPrimitive.createSimpleForm(owner,
                    args.getInputValue(ServiceContext.INP_FIRST_SERVICE_NAME),
target);
                container.appendChild(form);
                RuntimeAdapterHelper.createInputFields(form, missingArgs, args);
            }
        } catch (Exception ex) {
            Element errTxt = simpleText(owner, ex.getMessage(), "Error");
            container.appendChild(errTxt);
        }
    }

    /**
     * Check that arg has an value for all valid attributes.
     *
     * @param arg Arguments which holds the parameters
     * @exception Exception when required input parameters are missing
     */
    private String[] getMissingSearchArguments(Arguments args) {
        String[] missingArgs;
        Vector buffer = new Vector();
        for (int i = 0; i < SEARCH_ARGUMENTS.length; i++) {
            String argValue = args.getInputValue(SEARCH_ARGUMENTS[i]);
            if (argValue == null || argValue.length() == 0) {
                buffer.addElement(SEARCH_ARGUMENTS[i]);
            }
        }
        missingArgs = new String[buffer.size()];
        buffer.copyInto(missingArgs);
        return missingArgs;
    }

    /**
     * Search for users. <p>
     *
     * The service request contains the parameters.
     *
     * @param owner the owner document.
     * @param container    an element for the output result.
     * @param args    the service request holds the arguments for the user.
     */
    private void deleteUser(Document owner, Element container, Arguments
args) {[14]
        try {
            String userName = args.getInputValue(USER_NAME);
```

```
            if (userName != null && userName.length() > 0) {
                User user = modelServices.lookupUser(userName);      [15]
                user.delete();                                       [16]

                // Do not forget to save the changes
                modelFactory.save();

                Element successTxt = simpleText(owner, "User \"" + userName +
"\" was


removed!", "Success");
                container.appendChild(successTxt);
            } else {
                //construct the target parameters
                String target = RuntimeAdapterHelper.getURLPAoidParameter(args);
                target = target.concat("&" + COMMAND + "=" + DELETE_USER);
                Element form = PAPrimitive.createSimpleForm(owner,
                    args.getInputValue(ServiceContext.INP_FIRST_SERVICE_NAME),
target);
                container.appendChild(form);
                String[] missingArgs = new String[1];
                missingArgs[0] = userName;
                RuntimeAdapterHelper.createInputFields(form, missingArgs, args);
            }
        } catch (Exception ex) {
            Element errTxt = simpleText(owner, ex.getMessage(), "Error");
            container.appendChild(errTxt);
        }
    }

    /**
     * Add a single line text
     * @param text
     * @return Element with the text
     */
    private Element simpleText(Document owner, String text, String head) {
        Element simpleText = PAPrimitive.createSimpleText(owner);
        simpleText.appendChild(PAPrimitive.createSimpleTextItem(owner, head,
text));
        return simpleText;
    }

}
```

8

Using the Location API

This document describes the location application component API. Each section of this document presents a different topic. These sections include:

- Section 8.1, "Overview"
- Section 8.2, "Managing the Location Application Components"
- Section 8.3, "Using the Geocoder Interface"
- Section 8.4, "Using the Routing API"
- Section 8.7, "Using Business Directory Services"

8.1 Overview

The Wireless Edition provides a set of location application component APIs that enable developers to include geocoding, location marks, routing, and business directory (yellow pages) components into the Wireless Edition applications. Using these APIs, developers can integrate location services from geocoding, mapping, routing and yellow page third-party providers into Wireless Edition applications without having to write custom interfaces for each service.

The Wireless Edition framework allows developers to prioritize services based on quality, availability, or cost. The location application component APIs also include a wrapper function that maps location services providers to the APIs.

This chapter defines the Location APIs.

Note: The APIs described in this chapter define the interface between the service providers and the Wireless Edition developer. The APIs themselves mostly do not implement the core functionality. For more information on the APIs, refer to the Wireless Edition Java documentation.

8.2 Managing the Location Application Components

The Java class `SpatialManager` manages the following components:

- Geocoding Interface
- Point Interface
- Location Interface
- Mapper
- Router
- YPFinder

8.3 Using the Geocoder Interface

The Geocoder Interface provides the geographic location of a specified address using the `geocodeAddress` method. You enter the text for a geocoder in the same way that you enter the address on a standard letter. After you enter the address, the geocoding service translates it into geographic coordinates and expresses the location as a geographic point. In the following example, the Geocoder returns the coordinate of x:-122.262 / y:37.532 as the location of the given address.

firmName: "Oracle"

firstLine: "500 Oracle Parkway"

secondLine " "

lastLine: "Redwood City, CA 94065"

match mode: "tight"

Result: x:-122.262 / y:37.532

The Geocoder Interface is a required component for any spatial capability, including the Routing, Yellow Page, Mobile Positioner, Region, Mapper, or LocationMark

functions. The Geocoder Interface defines the top-level class that an application programmer accesses to use geocoding.

The Point Interface

The Geocoder API uses the Point objects to define the longitude and latitude of a point. The Point Interface uses the following methods:

- `getLabel`
- `setLabel`
- `getLatitude`
- `setLatitude`
- `getLongitude`
- `setLongitude`
- `getRadius`
- `setRadius`
- `getDistance`

The Location Interface

The Location Interface extends the Point Interface by defining a location with an address and longitude/latitude. The Location Interface uses the following methods:

- `getCompanyName`
- `setCompanyName`
- `getAddressLine1`
- `setAddressLine1`
- `getAddressLine2`
- `setAddressLine2`
- `getAddressLastLine`
- `setAddressLastLine`
- `getBlock`
- `setBlock`
- `getCity`

- setCity
- getCounty
- setCounty
- getState
- setState
- getPostalCode
- setPostalCode
- getPostalCodeExt
- setPostalCodeExt
- getCountry
- setCountry
- getMatchMode
- setMatchMode
- geocode

Note: Some providers may not be able to correctly identify some elements of the address, such as `getCity` or `getState` if the location object is constructed using `firstLine`, `secondLine`, and `lastLine`, because `lastLine` can contain city, state, and postal code formats that vary according to country and style.

8.3.1 Third-Party Geocoders

The Wireless Edition uses a Java interface to call the geocoders from the middle tier. The Wireless Edition API uses a ranked list of geocoders. Should the first of these geocoders fail, then the subsequent geocoders are used to return the search result. The failover is transparent to the user. An example of an XML configuration file for this API is as follows:

```
<?xml version="1.0" standalone="yes"?>
<Providers>
  <Provider
     ProviderName   ="MapQuest"
     ProviderTmpl   ="oracle.panama.spatial.geocoder.GeocoderImplMapqeust"
```

```
      URL            ="enterprise.mapquest.com"
      UserName       ="..."
      UserPassword   ="..."
      Parameters     = "..."/>
   <Provider>
      ProviderName   ="Vicinity"
      ProviderTmpl   ="orcale.panama.spatial.geocoder.GeocoderImplVicinity"
      URL            ="demo.vicinity.com"
      UserName       ="..."
      UserPassword   =""/>
    <Provider>
      ProviderName   ="MapInfoDirect"
      ProviderTmpl   ="orcale.panama.spatial.geocoder.GeocoderImplMapInfo"
      URL            ="http://..."
      UserName       ="..."
      UserPassword   =""/>
    <Provider>
      ProviderName   ="MapInfo"
      ProviderTmpl   ="orcale.panama.spatial.geocoder.GeocoderImplSQL"
      URL            ="http://..."
      UserName       ="..."
      UserPassword   =""/>
</Providers>
```

8.3.2 Reliability of Service

The Wireless Edition can use a set of third-party geocoders to ensure reliability. If one service fails, another acts as a backup. The Geocoding API fails if the entire set of external geocoders fail, or if access to the Web becomes unavailable. In the latter case, all Wireless Edition services, including the geocoders, become unavailable to the Wireless Edition. The Geocoder API automatically fails over if the geocoder becomes inaccessible over the Web, if it does not support addresses in the country where an address has been requested, or if it cannot find the requested address.

8.3.3 Ambiguous Address

If a user requests an address that cannot be identified by an exact location, the `GeoCodeResult` returns an array of location objects.

8.4 Using the Routing API

The Routing API provides driving directions and maps. This API is a standard interface between a routing service provider and the Wireless Edition developer. The Routing API includes route optimization methods to determine for example the shortest route and displays search results as a set of driving maneuvers. The Routing API can display search results as an overall map of the route as well as a map of the maneuvers themselves. In addition, the Routing API displays the geometry of the route (an array of points). The search results also include the distance of a specific driving maneuver and the total distance of the entire trip.

The Router Interface

The `Router` interface defines how an application programmer accesses the routing service. An object of a class implementing this interface is returned by the `SpatialManager`.

8.4.1 Source, Destination, and Via Points

The Routing API provides driving directions based on a source or starting point and a destination or end point. These points can be expressed as addresses or as longitudes and latitudes.

The Routing API may also include intermediate "via points". These via points are optional, and the API includes them when a traveler needs to start from point A, stop at points B and C, and then ends the journey at point D. In this case, point A is the source point, D is the destination point, and points B and C are the via points. Source, destination, and via points are the input parameters to the routing computation function described in Section 8.4.4, "Driving Distance".

8.4.2 Maneuver List

The Routing API expresses search results as a list of maneuvers or driving instructions. In addition, the Routing API describes the distance travelled during a single maneuver as well as the distance travelled up to, but not including, a certain maneuver. For example, the Routing API can return search results that include such instructions as "Turn RIGHT onto FRANKLIN ST." Maneuver information also includes a detailed maneuver map and a list of longitude and latitude coordinate points that correspond to each maneuver.

The Routing API uses the `Maneuver` class to define a single maneuver in a route.

8.4.3 Map Displays

The Routing API uses `requestMap` to include a map of the route. The Routing API provides two types of maps: an overview map (`getOverviewMap`) and a maneuver map (`getManueverMap`). These maps, which are returned as Java image objects or URLs, can be displayed in any size, scale, or zoom levels.

Note: The application may affect the size, scale, or zoom level of a map. In addition, some service providers may not support parameters for changing the size, scale, or zoom level of a map and may only support a single image format. Because different devices require different image formats, the application is responsible for image transformation between formats.

The Routing API uses `requestGeomery` to specify whether the exact geometry of the route is requested. This option, along with `requestMap`, can be disabled if the application only shows the list of maneuvers.

8.4.4 Driving Distance

The Router interface defines the top-level class that the application programmer accesses for routing. The Routing API uses `computeRoute` to perform routing computation based on source, destination, via points, and the routing options.

8.4.5 Routing Settings

The Routing API uses the `RoutingSettings` class to specify whether to generate a map or driving directions. Developers can specify the algorithms to choose the shortest distance from the source point to the destination point or the shortest driving time. The `RoutingSettings` class has two routing option types: basic and secondary. The basic options, which application developers specify in the constructor of a `RoutingSettings` object, include the following:

- Requests for maps.
- Requests for the geometry of a route.

Application developers can define the secondary options as either mandatory or preferred. The secondary options include the following:

- Optimization methods that compute the shortest driving time and shortest route.
- Map sizing functions.
- Route properties, such as *avoid toll roads*, *avoid ferry lanes*, and *avoid limited access highways*.

The secondary options can be set using `setSecondaryOption`. The first parameter is a `RoutingOption` object, which is a static constant defined in the `RoutingOption` class. It identifies the option for which a value is set. The second parameter is a `String` representing the value.

A secondary option is defined as mandatory by `setSecondaryOptionRequired`. The first parameter is the `RoutingOption` and the second parameter specifies whether this option requirement is mandatory. Unless this function is called, the default value is used.

The following defaults are applied if the application developer requests a secondary option without specifying whether it is mandatory or preferred:

- Optimization method: preferred
- Avoid Ferry: preferred
- Avoid Limited Access Hwy: preferred
- Avoid Toll: preferred
- Overview Map Size: mandatory
- Maneuver Map Size: mandatory
- Overview Map scale and zoom level: preferred
- Maneuver Map scale and zoom level: preferred

Note: If a secondary option is defined as mandatory but is not supported by a provider, the API automatically fails over to the next provider. If a secondary option is preferred but is not supported by a provider, the API does not check if other providers support the option.

The secondary class also includes the TSP (Traveling Salesman Problem) algorithm, `useTSP`. If `useTSP` is set to true, the routing provider is instructed to find a

near-optimal route that leaves the source and visits each via point once before arriving at the destination. The order in which the traveller visits the via points may be different from the order specified in the vector of via points. If `useTSP` is set to false, the order of the visit is the same as that specified in the vector of via points.

8.4.6 Routing Search Results

The Routing API uses `RoutingResult` class to define the output of a routing request. The routing result components include a list of maneuvers, the total distance of the trip, the total estimated driving time, and an overview map.

Multiple Language Support

If the routing provider supports multiple languages, then the Routing API selects a language based on the Java Locale object specified in the request to the router. The language setting can affect the maneuver narratives and the distance measures.

Routing XML File

An XML configuration file contains information about the routing service providers. For this example, the providers are MapInfo, MapBlast, and MapQuest.

```
<?xml version+"1.0" standalone="yes"?>
<providers>
  <Provider>
    ProviderName = "MapInfo"
    ProviderImpl = "oracle.panama.spatial.router.MapInfoRouterImpl"
    URL = "http://..."
    UserName = "..."
    UserPassword = "..."
    Parameters = "..."/>
Provider>
    ProviderName = "MapBlast"
    ProviderImpl = "oracle.panama.spatial.router.MapBlastRouterImpl"
    URL = "http://..."
    UserName = "..."
    UserPassword = "..."
    Parameters = "..."/>
Provider>
    ProviderName = "MapQuest"
    ProviderImpl = "oracle.panama.spatial.router.MapQuestRouterImpl"
    URL = "http://..."
    UserName = ""
    UserPassword = "..."
    Parameters = "..."/>
```

```
</Providers>
```

8.4.7 Display Functions

The Wireless Edition uses `Imagex` and `ImageFormats` classes to perform general operations on image files, such as maps. The `ImageFormats` class defines identifiers for various formats commonly used on wireless devices. The `ImageX` class enables the following operations:

- Conversion between image formats, such as **.gif**, **.bmp**, and **.wbmp**.
- Rotating an image by multiples of 90 degrees.
- Scaling an image to a larger or smaller target display.
- Flipping an image horizontally, vertically, diagonally and anti-diagonally.

The Imagex class enables the transformation of an image from one image format, such as **.gif**, into another format, such as **.wbmp**. For example, an external provider of a mapping service might support a **.gif** image while a particular device can only interpret **.wbmp** files. The Wireless Edition's ImageX API enables the creation of the **.wbmp** version of the image files.

In addition to image conversion, the `ImageX` class enables users to rotate images by 90 degrees for better viewing. For example, if a user retrieves a tall, vertical image that cannot be fully viewed on a wide display, the user can rotate the image by 90 degrees in either direction to fit it on a wide device display for complete viewing. The `ImageX` class enables images, such as maps, to be viewed in their entirety, rather than displaying them in several segments.

The `ImageX` class enables images to be scaled to fit the display of a device. Because transformation from **.gif** to **.wbmp** for display on a cell phone may also involve a scaling operation to make the image fit on the small display, the `ImageX` class enables scaling to be performed simultaneously with image format transformation. Combining scaling with image transformation enables developers to create several target images from one source (such as a 200x200 **.bmp** and a 100x50 **.wbmp** from the original 800x600 **.gif**), using one function call for each.

The flipping and rotation operations are not combined with image format transformation into a single operation, as these operations always result in **.bmp** target files. Rather than perform a rotation operation for several different target transformations, developers perform the rotation operation only once and then execute multiple format transformations from the **.bmp** target files. The quality of the results is not affected, while performance increases.

8.4.8 Mapping

The Wireless Edition uses the Mapper Interface to retrieve maps that display a list of marked points, addresses and business, a complete route, or a single maneuver.

In addition, the Mapper Interface enables users to adjust map images retrieved from third-party providers to any resolution and aspect ratio. In some cases, the requested map image may not display well on the target device. For example, if a user requests a map of California to display on a cell phone screen with a resolution of 300 (width) × 100 (height), the aspect ratio of the retrieved map of California may not match that of the cell phone's display screen. There are four solutions to this problem:

1. A map of size 50 (width) × 100 (height) is created and the rest of the cell phone screen remains empty. This is similar to taking a group photo vertically and then cutting the upper and lower sections off.
2. A map of size 300 (width) × 100 (height) is created but only a small segment -- 50 (width) × 100 (height) -- is covered by the area of interest (AOI). The remainder is a buffer to fill the screen. This is similar to taking a portrait photo horizontally, rather than vertically. To view the display properly, the user must turn the cell phone 90°.
3. A map of size 300 (width) × 600 (height) is created and cut in 6 segments measuring 300 (width) × 100 (height). This is similar to taking a group photo as a set of neighboring vertical segments and stitching them together.
4. A map of size 100 (width) × 300 (height) is created and a relatively large segment of size 100 (width) × 200 (height) is covered by the AOI. To receive a map of size 300 (width) × 100 (height), the original map is turned by 90°. The user must turn the cell phone accordingly to properly view the display.

In the Mapper Interface enables Solutions 2 and 3 separately, or in combination with Solution 4. The Mapper API describes the following functions:

- Returning single map images or matrices of map segments.
- Taking arrays of points, single points, routes or maneuvers to be mapped.
- Taking bounding boxes as parameters or inferring them from the marked points.

Each function can turn the image by 90° if doing so better matches the aspect ratio of the AOI and the visualization of the map image.

8.4.8.1 Mapping Routes and Maneuvers

Mapping of routes and maneuvers is more restricted than the mapping of points because maps of routes are actually provided by the routing component.

XML File

```
<?xml version="1.0" standalone="yes"?>
<Providers>
<Provider
  ProviderName  = "Vicinity"
  ProviderImpl  = "oracle.panama.spatial.mapper.MappingProviderVicinity"
  URL           = "http://…"
  UserName      ="..."
  UserPassword  = ""
  Parameters    = ""/>
<Provider
  ProviderName  = "Mapquest"
  ProviderImpl  = "oracle.panama.spatial.mapper.MapperImplMapquest"
  URL           = "…"
  UserName      = "…"
  UserPassword  = ""
  Parameters    = "…"/>
<Provider
  ProviderName  = "eLocation"
  ProviderImpl  = "oracle.panama.spatial.mapper.MappingProviderELocation"
  URL           = "http://…"
  UserName      =    ""
  UserPassword  =    ""
  Parameters    = ""/>
</Providers>
```

8.5 Finding a User's Location

The Mobile Positioner retrieves a mobile user's current physical location. The Mobile device positioning is performed by calling function `requestPosition()` in class `MobilePositioner`. This function takes user name, password and subscriber ID as input parameters. (You can set user name and password to null if they are not needed). The positioning result is encapsulated in a `PositionResult` object. Although different underlying positioning servers may return the position result in different formats, you can access them through the Wireless Edition interface. A `PositionResult` has an array of `PositionArea` objects. A `PositionArea` can be either a point location or a region and it has an accuracy level which indicates how precision of the positioning.

8.6 The Locationmark Interface

The form factor of some mobile devices limits the inputting and display of such spatial information as street address and location coordinates. To solve this problem, the Wireless Edition stores spatial information as a location mark, a name of a location that is meaningful to the user. For example, the location mark *My Home* identifies "123 Main Street, Somewhere City, CA 12345; Longitude = -122.42, Latitude = 37.58". Likewise, the location mark *Downtown San Francisco* corresponds to "The rectangle geometry within bounding points (Longitude/Latitude = -122.49, 37.79) and (Longitude/Latitude = -122.41, 37.74)."

Note: Location marks must be identified as a point for this release of Oracle9*i* Application Server Wireless Edition.

Locationmarks free users from inputting lengthy and complex alphanumeric strings on their mobile devices. Instead, users enter and manage the underlying spatial information to the locationmarks on a desktop computer. Users access this information by selecting the locationmark on their mobile device. The location marks are stored in the Wireless Edition repository. For more information on creating locationmarks, see *Oracle9i Application Server Wireless Edition Implementation Guide*.

Each user-defined locationmark is stored in the Wireless Edition repository. Each of these locationmark objects has a unique object ID registered in the object table. The value pair of the user name and locationmark name are used together as a composite key to identify the locationmark.

Geocoding provides the spatial information. The location marks are created using the `LocationMark` class.

8.7 Using Business Directory Services

Each external provider implements business directory, or yellow pages (YP) services differently; each provider has its own method of categorizing businesses. Some providers organize YP categories into a flat list, others organize them into a hierarchy tree. The hierarchies themselves may be deep or shallow, with a high or low fan-out, or they may be balanced or unbalanced.

To unify different hierarchy styles, those implementing the Wireless Edition must provide a custom hierarchy. Each node in the hierarchy can have references to a

category of one or more service providers. The Wireless Edition developer can customize the YP service to meet customer need by listing these provider service references. A Wireless Edition developer can rank YP services by popularity or by cost.

In searching for a business, a user may start traversing the hierarchy at its root, the default starting point. Alternatively, a user may enter a keyword that matches a starting point within the hierarchy.

8.7.1 Yellow Pages XML Files

Category Hierarchy Definition File

The customized hierarchy is represented as an XML file called the Category Hierarchy Definition File. Each category in this file can have any number of subcategories. There is no restriction on the number of nesting levels in the Category Hierarchy Definition File. Each node in the hierarchy tree can be linked to multiple business directory service content providers. An example of the Category Hierarchy Definition File is as follows:

```
<?xml version ="1.0" standalone = "yes"?>
<Categories>
  ...
  <Category
    CategoryName = "Berry crops">
    <Provider
      Name = "..."
      Parameter = "..."/>
      <Category
      CategoryName = "Cranberry farm">
      <Provider
        Name = "..."
        Parameter = "..."/>
      </Category>
      ...
      <Category
        CategoryName = "Ornamental nursery products">
        <Provider
          Name = "..."
            Parameter = "..."/>
          </Category>
             <Category
             CategoryName = "Florists'greens and flowers">
             <Provider
```

```
            Name = "..."
            Parameter = "..."/>
          </Category
        </Category
        <Category
          CategoryName = "Crops grown under cover">
          <Provider
            Name = "..."
            Parameter = "..."/>
        </Category>
      </Category>
      ...
    </Categories>
```

Provider Description File

The Provider Description File is an XML file that contains information about all the accessible business directory services content providers. An example of a Provider Description file is as follows:

```
<?xml version="1.0" standalone = "yes"?>

<Providers>
  <Provider
    ProviderName = "InfoUSALocal"
    ProviderImpl = "oracle.panama.spatial.yp.YPFinderImplInfoUSAlocal"
    URL = "jdbc:oracle:thin:@..."
    UserName = "..."
    UserPassword = "...">
   </Provider>
   <Provider
<Provider
    ProviderName = "InfoUSA"
    ProviderImpl = "oracle.panama.spatial.yp.YPFinderImplInfoUSA"
    URL = "..."
    UserName = "..."
    UserPassword = "...">
   </Provider>
   <Provider
<Provider
    ProviderName = "InfoUSALocal"
    ProviderImpl = "oracle.panama.spatial.yp.YPFinderImplMapquest"
    URL = "jdbc:oracle:thin:@..."
    UserName = "..."
    UserPassword = "...">
   </Provider>
```

```
   <Provider
<Provider
    ProviderName = "InfoUSALocal"
    ProviderImpl = "oracle.panama.spatial.yp.YPFinderImplVicinity"
    URL = "..."
    UserName = "..."
    UserPassword = "...">
   </Provider>
</Provider
```

8.7.2 Yellow Pages API

The yellow pages API provides application developers with the means to accommodate user search requests and search methods. For example, application developers list all businesses using `getBusinesses` in the YPFinder Interface. For searches starting at the root of the hierarchy, application developers use `getCategoryAtRoot` in the YPFinder interface. For searches that start from within the hierarchy using keywords, application developers use `getCategoryAtPath` in the YPFinder interface. Application developers can list the directly related business subcategories using `getSubCategories` in the YPCategory class. At the most specific, or leaf category level, application developers use `getBusiness` in the YPFinder simple interface to get a set of businesses. A business object of the YPBusiness class contains a name, location, telephone number, and description.

YPFinder Interface

The `YPFinder` interface defines how an application programmer accesses the business directory service. An object of a class implementing the interface is returned by the `SpatialManager`.

YPCategory Class

The `YPCategory` class defines a single category that is part of the hierarchy. This class lets users access businesses in the category and subcategories of the category.

9

Working with Wireless Edition XML

This document describes the XML formats that Wireless Edition uses to represent service content and its internal objects. Each section of this document presents a different topic. These sections include:

9.1 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.

9.2 Oracle XML Parser

The Wireless Edition uses the Oracle XML Parser for Java. The parser supports the W3C XML 1.0 Standard Recommendation. While the parser supports various character encoding standards, the Wireless Edition uses UTF-8 character encoding.

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 Wireless Edition 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`

9.3 Wireless Edition XML Formats

In Wireless Edition, 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

> **Note:** Wireless Edition XML is intended to be extensible. You can modify the DTDs that Wireless Edition provides, or create your own. This enables you to implement specialized functionality or to customize Wireless Edition object characteristics.

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 Wireless Edition 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.

Symbols

This document contains references for the Wireless Edition 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.

Table 9–1 DTD Symbols

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.

9.4 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:** Both standard adapter services based on the Web Integration adapter, and user-designed custom adapters can use the Adapter Result format.

The Wireless Edition does not specify an Adapter Result DTD. You can define the content any way you like. While the content is not validated by Wireless Edition, it must be well-formed. The only element Wireless Edition 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 Wireless Edition 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>
```

9.5 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.

Appendix A, "Simple Result DTD Reference" describes the content of the Simple Result DTD in detail.

9.6 The Wireless Edition Repository

The Wireless Edition repository holds the persistent Wireless Edition objects, such as end users, user groups, services, adapters, and transformers. Internally, Wireless Edition stores these objects as database objects in an Oracle8*i* 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 Wireless Edition. To use these utilities, you must understand the XML structure of the repository, as described in this section.

Wireless Edition 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 Section 9.9, "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 Wireless Edition 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.

9.7 Provisioning DTD

The Provisioning DTD defines the XML format for Wireless Edition users. You can use it to import users into the repository programmatically, or to import a large number of users at once. The Provisioning DTD declares only two elements:

- PUSR_LIST
- PUSR

PUSR_LIST Element

The Wireless Edition user list.

Note: `PUSR_LIST` is a subelement of `PanamaObjects`, the root provisioning element.

Subelement

PUSR+

PUSR Element

The PUSR element represents a Wireless Edition user. It includes the following 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 Wireless Edition 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, Wireless Edition 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 Wireless Edition 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.

9.8 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, Wireless Edition does not validate repository XML documents. Therefore, Wireless Edition 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`

`ADAP_LIST`

`AGEN_LIST`

`PSRV_LIST`

`TRAN_LIST`

`LDEV_LIST`

PGRP_LIST Element

The Wireless Edition group list.

Subelements

`PGRP`

PGRP Element

The `PGRP` element represents a Wireless Edition user group.

Attributes

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

PUSR_LIST Element

The Wireless Edition user list.

Subelements

PUSR

PUSR Element

The PUSR element represents a Wireless Edition 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, the Wireless Edition 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 Wireless Edition transformer list.

Subelements

XTRA

JTRA

XTRA Element

The XTRA element represents an XSLT Wireless Edition transformer.

Attributes

name
The name of the transformer.

Subelements

EXT_ATTR

EXT_ATTR Element

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

JTRA Element

The JTRA element represents a Wireless Edition Java transformer.

name
The name of the element.

className
The complete class name including attributes.

LDEV_LIST Element

The Wireless Edition logical device list.

Subelements

LDEV

LDEV Element

The LDEV element represents a Wireless Edition 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.

softkeys
The number of softkeys (used only for phones).

ScreenRows
The number of rows on the screen (useful for breaking text in pages).

ScreenColumns
The number of columns on the screen.

ScreenWidth
The width of the screen in points.

ScreenHeight
The height of the screen in points.

ADAP_LIST Element

The Wireless Edition adapter list.

Subelements

ADAP

ADAP Element

The ADAP element represents a Wireless Edition 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 Wireless Edition service object list.

Subelements

FOLD

LINK

MAST

BOMA

FOLD Element

The FOLD element represents a Wireless Edition 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 Wireless Edition 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.

areaId
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 Wireless Edition 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 Wireless Edition 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 Wireless Edition 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 Wireless Edition 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.

arealD
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 Wireless Edition 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 Wireless Edition user agent list.

Subelements

AGEN

AGEN Element

The AGEN element represents a Wireless Edition user agent.

address
The address of the user agent.

name
The name of the user agent.

user
The Wireless Edition user associated with the user agent.

logicalDevice
The logical device.

uniqueName
An identifying attribute set by Wireless Edition.

9.9 XML Tools

Wireless Edition includes the LoadXml tool and upload and download utilities for working directly with the XML interface. See the *Oracle9i Application Server Wireless Edition Configuration Guide* for more information.

A

Simple Result DTD Reference

The Simple Result DTD is the Wireless Edition 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 describes the content of the DTD including the elements, the usage, the attributes, the children, and the DTD declarations.

You can access the complete Simple Result DTD file, **simpleresult.dtd**, in the *Oracle_Home*/panama/dtd directory.

Each section of this document presents a different topic. These sections include:

Symbols

This document uses the following symbols to indicate the usage of subelements.

Table A–1 DTD Symbols

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.

A.1 SimpleResult

Element

SimpleResult is the result element of a Wireless Edition service request.

Usage

This element contains the actual content delivered to the end user.

Children

SimpleContainer

Attributes

Attribute	Description
name	The name of the element. This is optional.
link	A link for the element. This is optional.
title	The title of the element. This is optional.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

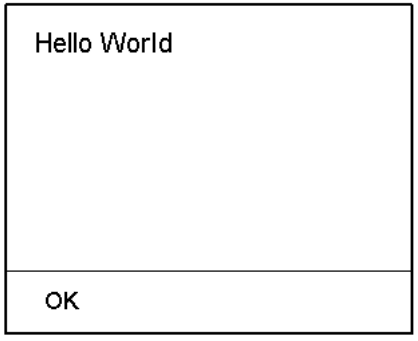
```
<!ELEMENT SimpleResult (SimpleContainer+)>

<!ATTLIST SimpleResult

title CDATA #IMPLIED

valign (top | center | bottom) "top"

halign (left | center | right) "left"

wrapmode (wrap | nowrap) #IMPLIED

name CDATA #IMPLIED

link CDATA #IMPLIED

 >
```

Example

```
<SimpleResult>
<SimpleContainer>

<SimpleText>
   <SimpleTextItem>Hello World</SimpleTextItem>
</SimpleText>

</SimpleContainer>
</SimpleResult
```

Example Output

Figure A–1 SimpleResult Example Output



A.2 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. This is the highest level tag.

Children

SimpleText

SimpleTimer

SimpleMenu

SimpleForm

SimpleTable

SimpleImage

SimpleBreak

SimpleHref

SimplePhone

SimpleEmail

SimpleHelp

Attributes

Attribute	Description
name	The name of the element. This is optional.
link	A link for the element. This is optional.
title	The title of the element. This is optional.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

<!ELEMENT SimpleContainer (SimpleHelp?, SimpleTimer?, (SimpleText | SimpleMenu | SimpleForm | SimpleTable | SimpleImage | SimpleBreak | SimpleHref | SimplePhone | SimpleEmail)+)>

<!ATTLIST SimpleContainer

valign (top | center | bottom) "top"

halign (left | center | right) "left"

wrapmode (wrap | nowrap) #IMPLIED

title CDATA #IMPLIED

name CDATA #IMPLIED

link CDATA #IMPLIED

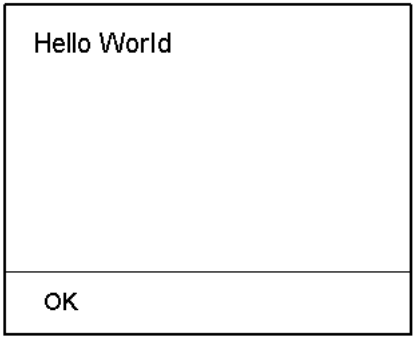
>

Example

```
<SimpleResult>
<SimpleContainer>
<SimpleText>
   <SimpleTextItem>Hello World</SimpleTextItem>
</SimpleText>

</SimpleContainer>
</SimpleResult>
```

Example Output

Figure A–2 SimpleContainer Example Output



A.3 SimpleText

Element

SimpleText is used for displaying one or more blocks of text.

Usage

Used for blocks of text which are defined by SimpleTextItem.

Children

SimpleTextItem

SimpleHelp

Attributes

Attribute	Description
name	The name of the element. This is optional.
link	A link for the element. This is optional.
title	The title of the element. This is optional.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.

Attribute	Description
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

```
<!ELEMENT SimpleText (SimpleHelp?, SimpleTextItem+)>
<!ATTLIST SimpleText
title CDATA #IMPLIED
name CDATA #IMPLIED
link CDATA #IMPLIED
valign (top | center | bottom) "top"
halign (left | center | right) "left"
wrapmode (wrap | nowrap) #IMPLIED
>
```

A.3.1 SimpleTextItem

Element

SimpleTextItem is a block of text.

Usage

Holds one block of text, normally a single paragraph.

Children

#PCDATA (the actual text)

Comment

There is no line feed at the end of each SimpleTextItem. See "SimpleBreak".

Attributes

Attribute	Description
name	The name of the element. This is optional.

Attribute	Description
link	A link for the element. This is optional.
title	The title of the element. This is optional.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

<!ELEMENT SimpleTextItem (#PCDATA)>

<!ATTLIST SimpleTextItem

name CDATA #IMPLIED

link CDATA #IMPLIED

title CDATA #IMPLIED

valign (top | center | bottom) "top"

halign (left | center | right) "left"

wrapmode (wrap | nowrap) #IMPLIED

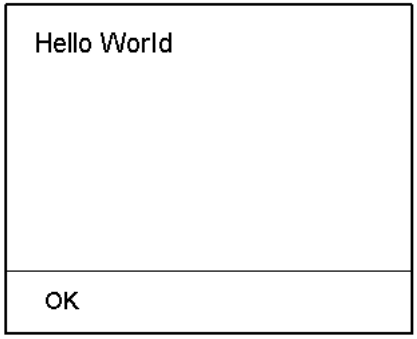
>

Example

```
<SimpleResult>
<SimpleContainer>

<SimpleText>
   <SimpleTextItem>Hello World</SimpleTextItem>
</SimpleText>

</SimpleContainer>
</SimpleResult
```

Example Output

Figure A–3 SimpleTextItem Example Output



A.4 SimpleMenu

Element

SimpleMenu displays a simple menu.

Usage

A simple menu with selectable menu items.

Children

SimpleMenuItem

SimpleHelp

Attributes

Attribute	Description
link	A link for the element. This is optional.
title	The title of the element. This is optional.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.

Attribute	Description
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

```
<!ELEMENT SimpleMenu (SimpleHelp?, SimpleMenuItem+)>
<!ATTLIST SimpleMenu
link CDATA #IMPLIED
title CDATA #IMPLIED
valign (top | center | bottom) "top"
halign (left | center | right) "left"
wrapmode (wrap | nowrap) #IMPLIED
>
```

A.4.1 SimpleMenuItem

Element

SimpleMenuItem is a single, selectable option in a menu.

Usage

The option may be another SimpleMenu element.

Children

SimpleMenu

Attributes

Attribute	Description
name	The name of the element. This is optional.
link	A link for the element. This is optional.
title	The title of the element. This is optional.
section	The section identifier within the WIDL. This is optional.

Attribute	Description
separator	A separator before or after the element. This is optional.
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.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

<!ELEMENT SimpleMenuItem (SimpleMenu?)>

<!ATTLIST SimpleMenuItem

name CDATA #IMPLIED

link CDATA #IMPLIED

title CDATA #IMPLIED

section CDATA #IMPLIED

separator (before | after | none) "none"

target CDATA #REQUIRED

icon CDATA #IMPLIED

valign (top | center | bottom) "top"

halign (left | center | right) "left"

wrapmode (wrap | nowrap) #IMPLIED

>

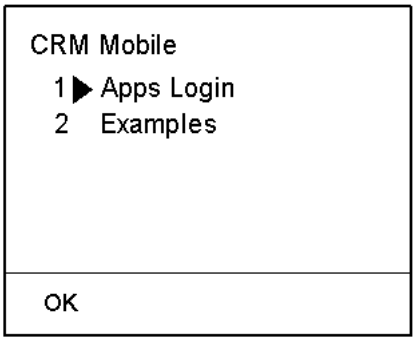
Example

```
<SimpleResult>
<SimpleContainer>

<SimpleMenu title="CRM Mobile">
   <SimpleMenuItem target="<%=baseUrl%>jtfwlgnm.jsp"
      title="Apps Login"> Apps Login</SimpleMenuItem>
   <SimpleMenuItem target="<%=baseUrl%>examples.jsp"
```

```
      title="Examples">Examples</SimpleMenuItem>
</SimpleMenu>

</SimpleContainer>
</SimpleResult
```

Example Output

Figure A–4 SimpleMenuItem Example Output



A.5 SimpleForm

Element

SimpleForm displays one or more input fields.

Usage

Used as a data-entry form.

Children

SimpleFormItem

SimpleHelp

SimpleFormSelect

Attributes

Attribute	Description
name	The name of the element. This is optional.
link	A link for the element. This is optional.
title	The title of the element. This is optional.
section	The section identifier within the WIDL. This is optional.
target	The link target for this form.
submit	The text for the submit button, default is "Submit".
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

```
<!ELEMENT SimpleForm (SimpleHelp?, (SimpleFormItem |
SimpleFormSelect)+)>
!ATTLIST SimpleForm
name CDATA #IMPLIED
link CDATA #IMPLIED
title CDATA #IMPLIED
section CDATA #IMPLIED
target CDATA #REQUIRED
submit CDATA #IMPLIED
valign (top | center | bottom) "top"
halign (left | center | right) "left"
wrapmode (wrap | nowrap) #IMPLIED
>
```

A.5.1 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)

SimpleFormSelect

Attributes

Attribute	Description
value	A value of the element.
default	A default value for optional fields, it is deprecated by the attribute of the element.
name	The name of the element. This is optional.
link	A link for the element. This is optional.
title	Specifies a title for the element, which may be used in the presentation of this object. This is optional.
mandatory	Indicates whether the form item is mandatory. Values are "yes" or "no". The default is "no".
maxLength	Specifies 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*.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

```
<!ELEMENT SimpleFormItem (#PCDATA)>
<!ATTLIST SimpleFormItem
name CDATA #IMPLIED
link CDATA #IMPLIED
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"
valign (top | center | bottom) "top"
halign (left | center | right) "left"
wrapmode (wrap | nowrap) #IMPLIED
>
```

A.5.2 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

Attribute	Description
name	The name of the element. This is optional.
link	A link for the element. This is optional.
title	The display name of the element. This is optional.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

<!ELEMENT SimpleFormSelect (SimpleHelp?, SimpleFormOption+)>

<!ATTLIST SimpleFormSelect

name CDATA #IMPLIED

link CDATA #IMPLIED

title CDATA #IMPLIED

valign (top | center | bottom) "top"

halign (left | center | right) "left"

wrapmode (wrap | nowrap) #IMPLIED

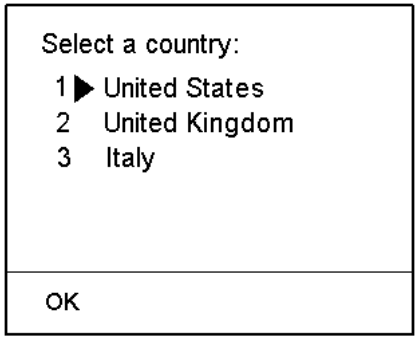
>

Example

```
<SimpleResult>
   <SimpleContainer>
      <SimpleForm target="<%baseUrl%>doCountry.jsp">
         <SimpleFormSelect name="country" title="Select a country:">
            <SimpleFormOption value="US">United States</SimpleFormOption>
            <SimpleFormOption value="UK">United Kingdom</SimpleFormOption>
            <SimpleFormOption value="IT">Italy</SimpleFormOption>
         </SimpleFormSelect>
      </SimpleForm>
   </SimpleContainer>
</SimpleResult>
```

Example Output

Figure A–5 SimpleFormSelect Example Output



A.5.3 SimpleFormOption

Element

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

Usage

A single option.

Children

#PCDATA (the text of the option)

Attributes

Attribute	Description
name	The name of the element. This is optional.
link	A link for the element. This is optional.
title	The display name of the element. This is optional.
value	The actual value, as character data, assigned when the user selects the option.

Attribute	Description
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

<!ELEMENT SimpleFormOption (#PCDATA)>

<!ATTLIST SimpleFormOption

name CDATA #IMPLIED

link CDATA #IMPLIED

title CDATA #IMPLIED

value CDATA #REQUIRED

valign (top | center | bottom) "top"

halign (left | center | right) "left"

wrapmode (wrap | nowrap) #IMPLIED

>

Example

```
<SimpleResult>
<SimpleContainer>

<SimpleForm title="Login"
   target="<%=baseURL%>jtfwvald.jsp"
   submit="Login">
   <SimpleFormItem name="username"
      title="Username: ">Username: </SimpleFormItem>
   <SimpleFormItem name="password"
      type="password"
      title="Password: ">Password: </SimpleFormItem>
</SimpleForm>

</SimpleContainer>
<?SimpleResult>
```

Example Output

Figure A–6 SimpleFormOption Example Output 1

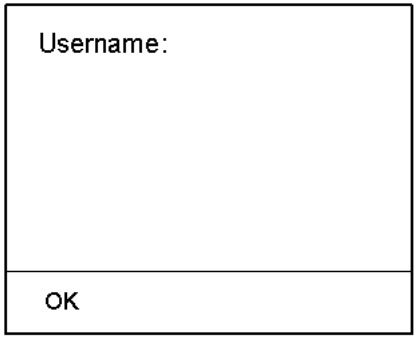
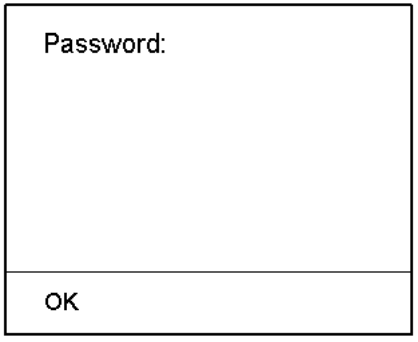


Figure A–7 SimpleFormOption Example Output 2



A.6 SimpleTable

Element

SimpleTable represents tabular data.

Usage

Used for simple tabular data only.

Children

SimpleTableHeader

SimpleTableBody

Attributes

Attribute	Description
name	The name of the element. This is optional.
link	A link for the element. This is optional.
title	The title for the table. This is optional.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

```
<!ELEMENT SimpleTable (SimpleTableHeader?, SimpleTableBody)>
<!ATTLIST SimpleTable
title CDATA #IMPLIED
name CDATA #IMPLIED
link CDATA #IMPLIED
valign (top | center | bottom) "top"
halign (left | center | right) "left"
wrapmode (wrap | nowrap) #IMPLIED
>
```

A.6.1 SimpleTableHeader

Element

SimpleTableHeader represents column headings.

Usage

Contains table header information, such as column headings.

Children

SimpleCol

Attributes

Attribute	Description
name	The name of the element. This is optional.
title	The title for the table. This is optional.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

```
<!ELEMENT SimpleTableHeader (SimpleCol+)>
<!ATTLIST SimpleTableHeader
title CDATA #IMPLIED
name CDATA #IMPLIED
valign (top | center | bottom) "top"
halign (left | center | right) "left"
wrapmode (wrap | nowrap) #IMPLIED
>
```

A.6.2 SimpleTableBody

Element

SimpleTableBody contains the actual tabular data.

Usage

Contains table data rows.

Children

SimpleRow

Attributes

Attribute	Description
name	The name of the element. This is optional.
link	A link for the element. This is optional.
title	The title for the table. This is optional.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

```
<!ELEMENT SimpleTableBody (SimpleRow+)>
<!ATTLIST SimpleTableBody
title CDATA #IMPLIED
name CDATA #IMPLIED
link CDATA #IMPLIED
valign (top | center | bottom) "top"
halign (left | center | right) "left"
wrapmode (wrap | nowrap) #IMPLIED
>
```

A.6.3 SimpleRow

Element

SimpleRow contains a single row of data.

Usage

Stores a table row.

Children

SimpleCol

Attributes

Attribute	Description
name	The name of the element. This is optional.
link	A link for the element. This is optional.
title	The title for the table. This is optional.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

```
<!ELEMENT SimpleRow (SimpleCol+)>
<!ATTLIST SimpleRow
title CDATA #IMPLIED
name CDATA #IMPLIED
link CDATA #IMPLIED
valign (top | center | bottom) "top"
halign (left | center | right) "left"
wrapmode (wrap | nowrap) #IMPLIED
>
```

A.6.4 SimpleCol

Element

SimpleCol represents a single table cell.

Usage

Stores a single value for a table cell.

Children

#PCDATA (the column value)

SimpleImage

SimpleHref

SimplePhone

Attributes

Attribute	Description
name	The name of the element. This is optional.
link	A link for the element. This is optional.
title	The title for the table. This is optional.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

<!ELEMENT SimpleCol (#PCDATA | SimpleImage | SimpleHref | SimplePhone)>

<!ATTLIST SimpleCol

title CDATA #IMPLIED

name CDATA #IMPLIED

link CDATA #IMPLIED

valign (top | center | bottom) "top"

halign (left | center | right) "left"

wrapmode (wrap | nowrap) #IMPLIED

>

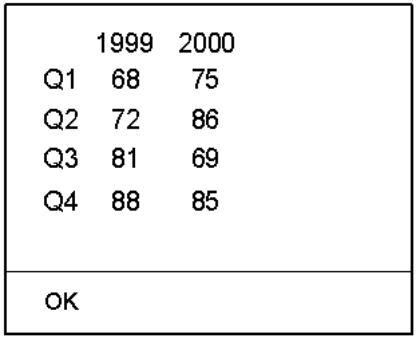
Example

```
<SimpleResult>
<SimpleContainer>
```

```
<SimpleTable title="Profitibility">

   <SimpleTableHeader>
      <SimpleCol> <SimpleCol>
      <SimpleCol>1999</SimpleCol>
      <SimpleCol>2000</SimpleCol>
   </SimpleTableHeader>

   <SimpleTableBody>
      <SimpleRow>
         <SimpleCol>Q1</SimpleCol>
         <SimpleCol>68</SimpleCol>
         <SimpleCol>75</SimpleCol>
       </SimpleRow>
      <SimpleRow>
         <SimpleCol>Q2</SimpleCol>
         <SimpleCol>72</SimpleCol>
         <SimpleCol>86</SimpleCol>
       </SimpleRow>
      <SimpleRow>
         <SimpleCol>Q3</SimpleCol>
         <SimpleCol>81</SimpleCol>
         <SimpleCol>69</SimpleCol>
      </SimpleRow>
      <SimpleRow>
         <SimpleCol>Q4</SimpleCol>
         <SimpleCol>88</SimpleCol>
         <SimpleCol>85</SimpleCol>
       </SimpleRow>
   </SimpleTableBody>

</SimpleTable>

</SimpleContainer>
</SimpleResult>
```

Example Output

Figure A–8 SimpleCol Example Output



A.7 SimpleImage

Element

SimpleImage displays an image.

Usage

Used to reference an external image.

Children

none

Attributes

Attribute	Description
target	The link target for the image.
src	The link for the image source.
alt	The alternative text, if the device does not support images.
width	The width of the image.
height	The height of the image.
valign	The vertical alignment: top, center, or bottom.

Attribute	Description
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.
addImageExtension	Specifies whether the stylesheet should add the image extension. The default is true (add image extension).

DTD Declaration

<!ELEMENT SimpleImage EMPTY)>

<!ATTLIST SimpleImage

src CDATA #REQUIRED

target CDATA #REQUIRED

alt CDATA #IMPLIED

width CDATA #IMPLIED

height CDATA #IMPLIED

valign (top | center | bottom) "top"

halign (left | center | right) "left"

wrapmode (wrap | nowrap) #IMPLIED

addImageExtension (true | false) "true"

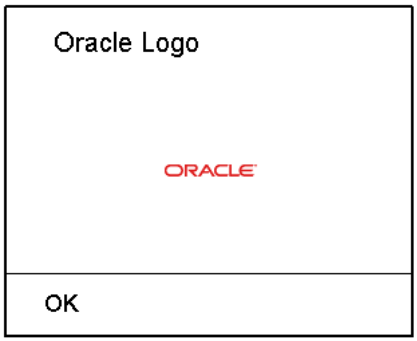
>

Example

```
<SimpleResult>
<SimpleContainer>

  <SimpleText>
    <SimpleTextItem>Oracle Logo</SimpleTextItem>
  </SimpleText>

 <SimpleImage target="<%=baseURL%>ora" alt="Oracle Logo">Oracle
Image</SimpleImage>

</SimpleContainer>
</SimpleResult>
```

Example Output

Figure A–9 SimpleImage Example Output



A.8 SimpleBreak

Element

SimpleBreak provides a logical break in the layout.

Usage

Used to provide a logical break.

Children

none

DTD Declaration

<!ELEMENT SimpleBreak EMPTY>

<!ATTLIST SimpleBreak>

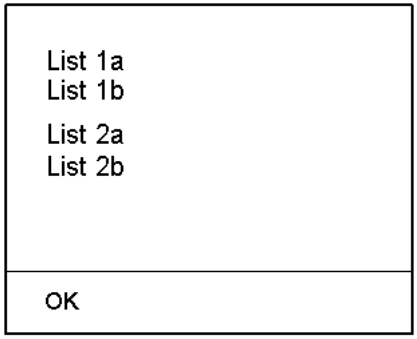
Example

```
<SimpleResult>
   <SimpleContainer>
      <SimpleText>
         <SimpleTextItem>List 1a</SimpleTextItem>
         <SimpleTextItem>List 1b</SimpleTextItem>
      </SimpleText>
      <SimpleBreak />
```

```
      <SimpleText>
         <SimpleTextItem>List 2a</SimpleTextItem>
         <SimpleTextItem>List 2b</SimpleTextItem>
      </SimpleText>
   </SimpleContainer>
</SimpleResult>
```

Example Output

Figure A–10 SimpleBreak Example Output



A.9 SimplePhone

Element

SimplePhone references a phone number.

Usage

A phone number.

Children

#PCDATA (the name of the phone)

Attributes

Attribute	Description
target	A phone number.

Attribute	Description
name	The name of the element. This is optional.
link	A link for the element. This is optional.
title	The title for the table. This is optional.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

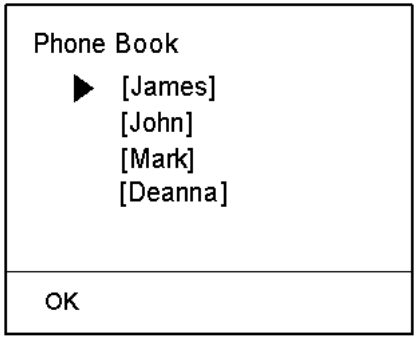
```
<!ELEMENT SimplePhone (#PCDATA)>
<!ATTLIST SimplePhone
target CDATA #REQUIRED
title CDATA #IMPLIED
name CDATA #IMPLIED
link CDATA #IMPLIED
valign (top | center | bottom) "top"
halign (left | center | right) "left"
wrapmode (wrap | nowrap) #IMPLIED
>
```

Example

```
<SimpleResult>
  <SimpleContainer>
    <SimpleText>
      <SimpleTextItem>Phone Book</SimpleTextItem>
    </SimpleText>
    <SimplePhone target="14155551212">James</SimplePhone>
    <SimplePhone target="16505551212">John</SimplePhone>
    <SimplePhone target="14085551212">Mark</SimplePhone>
    <SimplePhone target="17075551212">Deanna</SimplePhone>
  </SimpleContainer>
</SimpleResult>
```

Example Output

Figure A–11 SimplePhone Example Output



A.10 SimpleEmail

Element

SimpleEmail references an email address.

Usage

Used to reference an email address.

Children

#PCDATA (the name of the user)

Attributes

Attribute	Description
target	The email address.
name	The name of the element. This is optional.
link	A link for the element. This is optional.
title	The title for the table. This is optional.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.

Attribute	Description
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

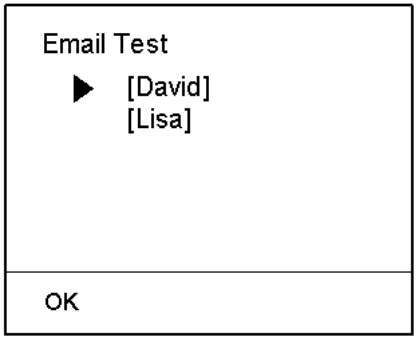
```
<!ELEMENT SimpleEmail (#PCDATA)>

<!ATTLIST SimpleEmail

target CDATA #REQUIRED

title CDATA #IMPLIED

name CDATA #IMPLIED

link CDATA #IMPLIED

valign (top | center | bottom) "top"

halign (left | center | right) "left"

wrapmode (wrap | nowrap) #IMPLIED

>
```

Example

```
<SimpleResult>
<SimpleContainer>
    <SimpleText>
       <SimpleTextItem>Email Test</SimpleTextItem>
    </SimpleText>
    <SimpleEmail target="david123@oracle.com">David</SimpleEmail>
    <SimpleEmail target="lisa123@oracle.com">Lisa</SimpleEmail>
</SimpleContainer>
</SimpleResult>
```

Example Output

Figure A–12 SimpleEmail Example Output



A.11 SimpleHref

Element

SimpleHref provides an anchor to another resource.

Usage

An anchor to another resource.

Children

#PCDATA (the name of the link)

Attributes

Attribute	Description
target	The link target for the reference.
name	The name of the SimpleHref, used to override button actions.
link	A link for the element. This is optional.
title	The title for the table. This is optional.
type	Specifies the type of SimpleHref, used to bind the target to a device-specific button.

Attribute	Description
label	Specifies the label to use, if SimpleHref is bound to a button.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

```
<!ELEMENT SimpleHref (#PCDATA)>
<!ATTLIST SimpleHref
target CDATA #REQUIRED
title CDATA #IMPLIED
name CDATA #IMPLIED
link CDATA #IMPLIED
type (accept | option1 | option2)
label CDATA
valign (top | center | bottom) "top"
halign (left | center | right) "left"
wrapmode (wrap | nowrap) #IMPLIED
>
```

Example

See Section A.12, "SimpleHelp".

Example Output

See Section A.12, "SimpleHelp".

A.12 SimpleHelp

Element

SimpleHelp indicates a help text.

Usage
Used for help text.

Children
#PCDATA (the name of the help text)

Attributes

Attribute	Description
name	The name of the SimpleHref, used to override button actions.
link	A link for the element. This is optional.
title	The title for the table. This is optional.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration
<!ELEMENT SimpleHelp (#PCDATA)>

<!ATTLIST SimpleHelp

title CDATA #IMPLIED

name CDATA #IMPLIED

link CDATA #IMPLIED

valign (top | center | bottom) "top"

halign (left | center | right) "left"

wrapmode (wrap | nowrap) #IMPLIED

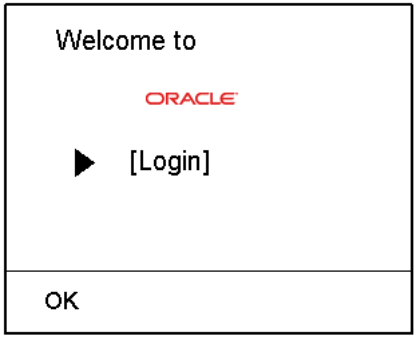
>

Example
```
<SimpleResult>
<SimpleContainer>

  <SimpleText>
```

```
    <SimpleTextItem>Welcome to</SimpleTextItem>
  </SimpleText>

  <SimpleImage target="<%=baseURL%>ora" alt="Oracle Logo">Oracle
Image</SimpleImage>
  <SimpleBreak></SimpleBreak>
  <SimpleBreak></SimpleBreak>

  <SimpleHref target="<%=baseURL%>login.jsp">Login</SimpleHref>

</SimpleContainer>
</SimpleResult>
```

Example Output

Figure A–13 SimpleHelp Example Output



A.13 SimpleTimer

Element

Defines a timer for displaying a SimpleContainer.

Usage

When the timer expires the target url is displayed.

Children

none

Attributes

Attribute	Description
timer	The numeric timer value in 1/10 of a second.
target	The URL to be executed when the timer has expired.
name	The name of the SimpleHref, used to override button actions.
link	A link for the element. This is optional.
title	The title for the table. This is optional.
valign	The vertical alignment: top, center, or bottom.
halign	The horizontal alignment: left, center, or right.
wrapmode	The wrap mode: wrap or nowrap.

DTD Declaration

<!ELEMENT SimpleTimer EMPTY>

<!ATTLIST SimpleTimer

timer %number #REQUIRED

title CDATA #IMPLIED

name CDATA #IMPLIED

link CDATA #IMPLIED

target CDATA #REQUIRED

valign (top | center | bottom) "top"

halign (left | center | right) "left"

wrapmode (wrap | nowrap) #IMPLIED

>

Example

```
<SimpleResult>
  <SimpleContainer name = "Advertisement Container">
  <SimpleTimer target ="#actualpage" timer = "30" />
    <Simple Text>
      <SimpleTextItem> This page can contain advertisement </SimpleTextItem>
```

```
    </SimpleText>
  </SimpleContainer>

  <SimpleContainer name ="actualpage">
    <SimpleText>
      <SimpleTextItem> Welcome to the actual page </SimpleTextItem>
    </SimpleText>
  </SimpleContainer>
</SimpleResult>
```

B

Runtime System Variables

This document describes the system parameters that are recognized by the Wireless Edition Runtime application. The values of some of these parameters can be referred to explicitly in the result XML as macros.

Each section of this document presents a different topic. These sections include:

- Section B.1, "System Parameters"
- Section B.2, "Macros"

B.1 System Parameters

The system parameters that are named here are processed by the Wireless Edition Runtime application. Developers can use these parameter names, which are included in the PARAMETERS AttributeCategory in Request, when constructing a URL request.

When these parameter names appear in the code, they should have two underscore characters before and two underscore characters after each parameter name. The system looks for these underscore characters when it parses the code for the parameter names.

Table B–1 Runtime System Defined Request Parameters

Parameter Name	Possible Parameter Value	Description	Example
PAdebug	Any non-null value	Enables the debug option for the current request.	`www.oracle.com/ptg /rm?PAservicepath=/myservice /finance& PAsection=getQuote&PAdebug=true`

Parameter Name	Possible Parameter Value	Description	Example
PAuserid	User's name	The name of the user	`PAuserid=Scott`
PApassword	User's password.	Specifies the user's password duing session login.	`PApassword=tiger`
PAoid	The oid of a service.	Referring a service via its OID.	`PAoid=1234`
PAsection	The name of a PAsection.	The current WIDL adapter supports multiple sections, PAsection identifies which section to run.	`www.oracle.com/ptg/rm?PAservicepath=/myservice/finance&` `PAsection=getQuote`
PAx	Result transformer name.	Specifies the result transformer. The default is the result transformer currently in use by the PAsection parameter.	`PAx=getQuote`
PAlogin	Any non-null value.	Logs in to the current session.	`www.oracle.com/ptg/rm?PAlogin=true`
PAlogoff	Any non-null value.	Logs out of the current session.	`www.oracle.com/ptg/rm?PAlogoff=true`
PAsid	This value is internally generated.	The session ID.	`PAsid=XXXXX`

Parameter Name	Possible Parameter Value	Description	Example

PAgetPage	A SimpleResult XML page.	Creates a URL to test how a SimpleResult XML is converted to the target markup language.
		
		You can imbed this parameter in any service invocation URL to return the transformed result of the given SimpleResult DTD rather than the invoked service.
			`www.oracle.com/ptg/rm?PAoid=123&PAgetPage=<SimpleResult><SimpleText><SimpleTextItem>`
			`HelloWorld`
			`</SimpleTextItem></SimpleText></SimpleResult>`

B.2 Macros

The result XML can refer to some system-defined parameters using Runtime system variables (macros) which are substituted by the PostProcessor before the final result is returned to the requester.

When these macros appear in the code, they have two underscore characters before and two underscore characters after each macro name. The system looks for these underscore characters when it parses the code for the parameter names.

Table B–2 Runtime Macros

System Value Macro	Description
"_LOGICAL_DEVICE"	The name of the logical device from which the request is being initiated
"_REQUEST_NAME"	The request name

System Value Macro	Description
__SESSION__	The URL to refer to the current session.
__SERVICE_NAME__(*)	The name of the service being invoked by the request.
__SERVICE_NAME_ENC__(*)	The encoded service path of the service being invoked by the request.
__SR_FROM_ADAPTER__	The service request object in XML form after the adapter invocating, but before applying the result transformer if any.
__SR_FROM_SERVICE__	The service request object in XML form after the adapter invocating and applying the result transformer if any.
__UserLanguage__	The accepted languages by the requesting device, can be null.
__PAuserid__	The name of the user.
__PAservicepath__	Referring a service via its service path.
__PAoid__	Referring a service via its OID.
__PAsection__	The current WIDL adapter supports multiple sections, PAsection is used to identify which section to run.
__userRef__	The oid of the user who is making the request.
__%PARAM_NAME%__	Where %PARAM_NAME% is the parameter name including both user and system parameters used in the query string of the URL.

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 Wireless Edition 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 Wireless Edition processing.

core

The Wireless Edition component that manages the Wireless Edition repository and service requests.

device portal

The interface where mobile device users access their Wireless Edition 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 Wireless Edition 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.

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 Wireless Edition repository.

public service tree

A service tree that is owned by the Wireless Edition system, and can be accessed by any end-user.

repository

An Oracle8*i* database which stores all Wireless Edition objects, such as users, groups, adapters, and services.

request

A query to initiate a desired Wireless Edition service. Requests are submitted on behalf of end-users to the Wireless Edition server.

request manager

The Wireless Edition component that processes requests for services. The request manager authenticates the user, submits the request to the Wireless Edition 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 Wireless Edition repository, which includes pre-built objects such as transformers, adapters, and logical devices.

service

A core object used in a Wireless Edition server to represent a unit of information requested by, and delivered to, a Wireless Edition 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 Wireless Edition 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. Wireless Edition 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 Wireless Edition repository. Thin HTML does not include support for frames, JavaScript, or other advanced features.

transformer

A Wireless Edition object that converts content returned by Wireless Edition 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 Wireless Edition 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 Wireless Edition 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 Wireless Edition service.

Wireless Edition XML

A set of DTDs and XML document conventions used by Wireless Edition to define content and internal objects.

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

M

N

O

P

R

S

T

U

X