Oracle9*i*AS Wireless

Developer's Guide

Release 2 (9.0.2)

May 2002

Part No. A90485-02

ORACLE

Oracle9*i*AS Wireless Developer's Guide, Release 2 (9.0.2)

Part No. A90485-02

Contents

5 Filling Out Forms for Data Entry and Navigation

6 Advanced User Interactions and Channel Optimization

7 Mobile Modules

8 XML Tag Glossary

Part III Oracle9*i*AS Wireless Platform and Services

9 Mobile Service Developer's Tools

10 Core Technologies

11 Advanced Customization

12 Alert Engine and Data Feeds

13 Push Service and SMS

19 m-Commerce

20 Location-Based Module

Index

Send Us Your Comments

Oracle9*i*AS Wireless Developer's Guide, Release 2 (9.0.2)

Part No. A90485-02

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

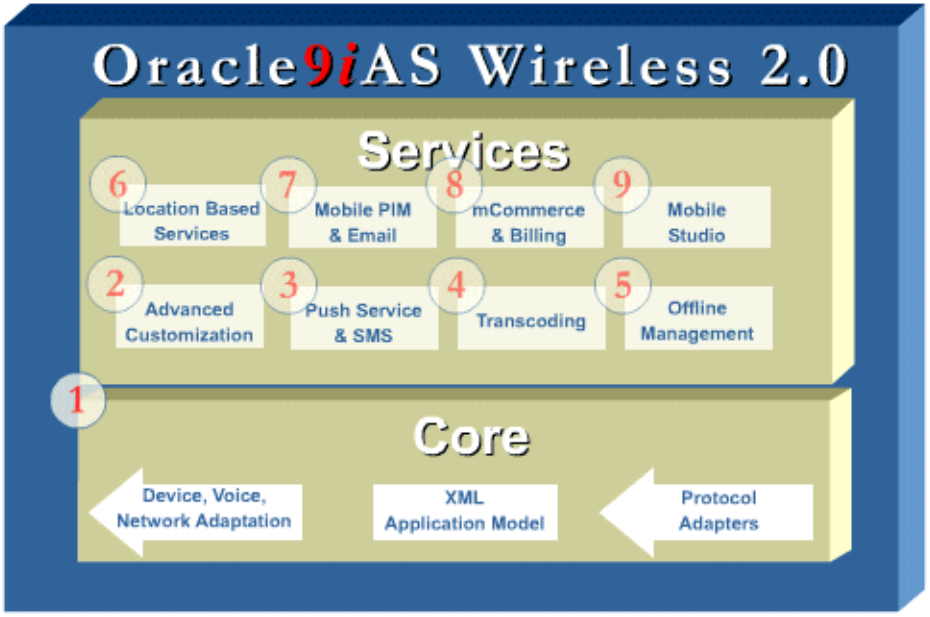
If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

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

Preface

This guide discusses how you can use Oracle9*i*AS Wireless to develop and deliver mobile services to any mobile device. The structure of this document follows the architecture of Oracle9*i*AS Wireless.

Figure 1. Oracle9iAS Wireless Architecture



This Guide includes the following Parts and Chapters:

Section	Content
Part I, "Introduction"	
Chapter 1, "Introduction"	Overview of Oracle9*i*AS Wireless
Part II, "Oracle9iAS Wireless XML Developer's Guide"	
Chapter 2, "XML Overview"	Overview of XML.
Chapter 3, "Displaying and Formatting Content"	Sample applications and methods for formatting and displaying XML content.
Chapter 4, "Application Navigation"	Navigating and linking among XML content.
Chapter 5, "Filling Out Forms for Data Entry and Navigation"	Using forms to simplify filling out and navigating XML content.
Chapter 6, "Advanced User Interactions and Channel Optimization"	Advanced user interactions with XML content.
Chapter 7, "Mobile Modules"	Using XML to develop and deploy mobile modules.
Chapter 8, "XML Tag Glossary"	Abstract device markup language used in the OracleMobile Online Studio application framework.
Part III, "Oracle9iAS Wireless Platform and Services"	
Chapter 9, "Mobile Service Developer's Tools"	Building applications using Oracle9*i*AS Wireless.
Chapter 10, "Core Technologies"	Core technologies used by Oracle9*i*AS Wireless.
Chapter 11, "Advanced Customization"	Adapting applications to increase mobile application efficiency.
Chapter 12, "Alert Engine and Data Feeds"	Publish timely information for subscribers from a variety of data sources.
Chapter 13, "Push Service and SMS"	Push and SMS Services architectures, and how to use these services to create and deploy mobile applications.
Chapter 14, "Transcoding"	Reformatting device / markup language for use on any web-enabled device.
Chapter 15, "Using Location Services"	Specialized services for developing location-based applications.
Chapter 16, "Offline Management"	Using Oracle9*i* Lite for offline management of content.
Chapter 17, "Mobile Studio"	Using Oracle Mobile Studio to develop wireless applications.

Section	Content
Part IV, "Oracle9iAS Wireless Modules"	
Chapter 18, "Mobile PIM and eMail"	Integrating PIM and e-mail services into your mobile applications.
Chapter 19, "m-Commerce"	Integrating m-Commerce and Billing services into your mobile applications.
Chapter 20, "Location-Based Module"	Integrating Location-Based services into your mobile applications.
"Index"	Index.

Intended Audience

This Guide is intended for developers of wireless applications.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Related Documents

Here is a partial list of related documents that will provide you with important information concerning Oracle9*i*AS Wireless and related products/components:

- *Oracle9iAS Wireless Getting Started and System Guide*—all the information you need to be up and running in the shortest possible time.
- *Oracle9iAS Wireless Release Notes*—final notes about the products, since the Documentation and Help were produced.
- Oracle9*i*AS Wireless online Help (included in the product)
- Javadoc with sample code included in product directory structure
- Oracle9*i*AS documentation (HTML and PDF library)
- Oracle Technology Network

  http://otn.oracle.com

  Oracle Technology Network is your main resource for information, samples, updates, and other downloads for your products. Stylesheets, drivers, documentation updates, sample code, demonstration software, and other valuable resources are available to you on OTN. Sign-up (if you haven't already done so; it's free!) with OTN to gain access and receive up-to-the-minute information about Oracle products and practices.

Part I

Introduction

Part I contains introductory information about Oracle9*i*AS Wireless.

- Chapter 1, "Introduction"

1 Introduction

This document provides an overview of Oracle9*i*AS Wireless. Each section of this document presents a different topic. These sections include:

- Section 1.1, "Overview"
- Section 1.2, "Wireless Internet Components"
- Section 1.3, "Developing Mobile Internet Applications"
- Section 1.4, "Oracle9iAS WirelessArchitecture"
- Section 1.5, "Oracle9iAS WirelessCore and Services"

1.1 Overview

Oracle9*i*AS Wirelessenables enterprises to deliver any content or application to any device with any protocol across any wireless network.

Using Wireless, you need only write an application once to have it run on any type of device. Wireless adapts information from any content source into an open XML format and then transforms the content into any markup language supported by any wireless device. Additionally, Wireless includes many advanced services such as location-based services, secure mobile commerce, and push services via SMS, WAP-Push and e-mail.

1.2 Wireless Internet Components

There are many infrastructure components that work together to make the wireless Internet function. The components include:

Wireless Devices and Microbrowsers

The user accesses the Internet using a Wireless Internet device - this device typically runs a microbrowser. (This is analogous in the fixed Internet world to a personal computer running a standard Internet browser). Wireless devices also include in-car systems and voice technology to access information from a traditional phone.

Wireless Markup Language

Each Wireless Device also speaks a language called a markup language - the markup language specifies how information should be presented on the device. Common Markup Languages include VoiceXML, WML, and HDML.

Protocols

Protocol is the method that is used to deliver the content data to the devices.

1.2.1 The Wireless Network

Wireless separates the concerns of the wireless network from developers, greatly simplifying the development and management of wireless applications.

1.2.1.1 Networks

Networks are the underlying infrastructure that is used by the wireless carriers. A large factor of networks is the bandwidth and the connection type. For example, 2.5G and third generation networks will provide high-speed access and always-on capabilities.

1.2.1.2 Wireless Gateways

Wireless Devices speak a variety of protocols such as WAP (Wireless Access Protocol), SMS (Short Messaging Service), Voice and others. The Wireless Gateway translates the wireless protocol request to the standard HTTP protocol. Note that Wireless Protocols are more efficient over the Wireless Networks than the standard HTTP protocol - this is one of the primary reasons that Wireless Internet clients do not speak HTTP directly.

1.2.1.3 Wireless Services

Wireless Services have a wide variety of forms including database information, personalization, alerts, and location services. The large number of content sources adds to the complexity of having a manageable way to deliver each application to every type of device in the most optimized fashion.

1.2.1.4 Application Servers

Application servers have come into play to increase the efficiency of application development, deployment, and management. The Wireless Application Server connects the wireless content source over the wireless network to the wireless Gateway or Device. To do so, it adapts the content from the content source, personalizes it for individual users, and converts (or transforms) it to the specific markup language spoken by the wireless device being used.

1.3 Developing Mobile Internet Applications

Wireless application development is typically constrained by limitations in user input, device display form factor, and the several different wireless device standards currently in use.

1.3.1 User Input Limitations

The keypads of cellular phones limit the user's ability to enter lengthy alphanumeric strings. The limited data entry capability makes cellular phones and other hand-held devices better suited to carry out specific inquiries and transactions rather than for Internet surfing.

1.3.1.1 Device Display Form Factor

The screen size and display capability of devices vary tremendously - since the Internet is likely to be accessed from a variety of different devices, it is not practical to optimize every application for every single device available. The requirements that these two limitations place on a Wireless Platform are twofold: first, the platform must be able to exploit device-specific functionality such as voice browsing which make it easier to navigate through Internet services from a wireless device; and second, the platform must provide ways to find and execute Internet services quickly and effectively by personalizing services and content to make them relevant to individual users. The user experience is far richer and more effective when the Wireless Platform supports a variety of personalization facilities such as allowing users to personalize which services they see, to see different information based on the device they are using, and to see different information based on the geographic location they are accessing the Internet from.

1.3.2 Myriad Wireless Device Standards

Even though wireless Internet standards are emerging, there are still diverse standards supported by wireless devices. Each device speaks a different wireless

protocol and supports a variety of different Wireless Markup Languages – these different standards preclude a developer from writing every application to individually support every single device available. As a result, as companies must choose a software platform that will radically simplify how they develop and deploy mobile portals and Internet applications.

To enable cross-platform support, a wireless Internet software platform must meet two requirements:

1.3.2.1 Support a Broad Variety of Devices and Protocols

First, the wireless software platform should be able to support a broad variety of wireless devices, each of which supports a different markup language, a different microbrowser, and a different communication protocol. Most Wireless Platforms support only the WAP Protocol and as a result, applications built to such a platform cannot be accessed from an i-Mode, Voice, SMS or a Blackberry device.

1.3.2.2 Write Applications Once and Deliver Them Anywhere

Web site developers must develop a Web site for each wireless type of device. Because of the increasing variety of wireless devices, such an application development paradigm does not scale; a developer must be able to develop a Web site once and have the software platform deliver it to any wireless device.

1.3.3 Heterogeneous Sources of Content

In addition to wireless devices, application development and deployment can also be complicated by the fact that the wireless Internet does not require that content or applications be specific to it.

1.3.3.1 Leveraging Existing Content

To leverage existing Internet content and e-Business applications for the wireless environment, the software platform must be able to easily re-use any Internet content or application, no matter how it was originally built to a wireless device. This requires the platform to be able to "adapt" content from a variety of repositories whether it comes from an Internet web site, from an e-Mail server, or from a database. Additionally, the wireless platform must also be able to adapt any Internet content whether the application has been built using Java, Visual Basic, PERL, PL/SQL, PHP, server-side scripting, or any other web site development language.

The wireless software platform must be able to support web sites or Internet applications that are developed specifically for the wireless Internet. It must

provide a seamless set of facilities to develop such web sites using open standards such as Java and XML.

1.3.3.2 Application Performance and Scalability Requirements

A Wireless Internet Platform must also be able to meet scalability requirements in three ways:

1.3.3.2.1 Support a Large Numbers of Users Since Wireless Carriers and Portals support Millions of subscribers, the Wireless Internet Platform must provide facilities to centrally manage these users, their security and access control privileges, and their ability to personalize services.

1.3.3.2.2 Support a Large Number of Concurrent Sessions Additionally, users who access the Internet from wireless devices typically have relatively long- lived conversational interactions with a number of different web services, in addition users desire always-on capabilities for fast notification of messages. Due to the limited bandwidth on the wireless network and the lack of support for "cookies" in most wireless gateways, it is not possible to push the user's session state from the server to the client. As a result, a wireless Internet platform must be able to manage user sessions and maintain session state in a highly scalable fashion.

1.3.3.2.3 Support a Large Volumes of Content Finally, since wireless users access large volumes of content which needs to be delivered very efficiently to their client, the Wireless platform must be able to use caching and share data facilities to serve large volumes of content efficiently.

1.3.3.3 Evolving Wireless Internet Market Requirements

Wireless Internet users want to send messages, browse information and services, carry out wireless commerce transactions and run complicated business applications. Most wireless software platforms only address a small set of requirements requiring users to choose one wireless software infrastructure for messaging, another to browse content, and yet another for mobile commerce. It is critical that a wireless platform must seamlessly integrate facilities for wireless messaging, content browsing, voice access, mobile commerce and business applications to allow developers to combine these facilities in building state-of-the-art applications and portals.

1.3.3.4 Evolving Wireless Standards

Wireless standards are evolving rapidly. At the network level standards such as CDMA, GSM, TDMA, iDEN, SMS, i-Mode, GPRS, and UMTS are all evolving; at the device level, standards such as VoxML and VoiceXML promise to change how the wireless Internet is used. As a result of these differences, a wireless Internet Platform must meet two requirements: first, it must be current with wireless standards such as i- Mode, WAP, SMS, GPRS, 3G and others; and second, it must support open industry standards such as XML, XHTML, Java Servlets, Java Server Pages for application development. Wireless address these issues making a complete wireless solution for businesses.

1.4 Oracle9*i*AS WirelessArchitecture

Wireless is the mobile component of the Oracle9*i*Application Server, an integrated suite for internet-enabling your applications and portals. Oracle9*iAS* sits on the Oracle9*i* Database, which is used as the secure repository for all the components. Oracle9*iAS* runs the Oracle e-Business suite as well as partner applications.

The Oracle9*iAS* is Oracle's comprehensive and integrated application server. It runs any Web site, portal, or Internet application and makes your Web site and all your applications accessible from any browser or mobile device. You can deliver tailored 1:1 customer experiences through real-time personalization and satisfy demands for current business information using Oracle9*iAS* integrated business intelligence services. You can simplify your management tasks by using the single management console provided with Oracle9*iAS*.

Wireless simplifies wireless development and deployment by providing the ability to deliver any content with any device, any protocol and across any Wireless Network with the core. Wireless leverages open standards such as XML, Apache and J2EE, to deliver a high performance, scalable wireless infrastructure.

1.4.1 Mobile Services

Built on Wireless are Mobile Services such as PIM (Personal Information Management), Email, and Location Based Services that simplify wireless enabling applications and portals. These are reusable application components that increase the time to market of mobile applications. The services can be configured out of the box or extended for custom abilities.

1.4.2 Processing a Request for a Wireless Service

Wireless processes a request for a wireless service as follows:

1. Sending a Wireless Request
2. Recognizing and Authenticating the Wireless Device
3. Establishing the Wireless Session
4. Translating the request over the Internet
5. Connecting to the Application Server
6. Recognizing the User's Information
7. Processing the Wireless Request

1.4.2.1 Sending a Wireless Request

A user invokes a Wireless service from a wireless Internet device by dialing the telephone number for the appropriate service provider. The microbrowser on the wireless device sends a request to the wireless network base station. The request can be sent over a variety of different protocols, such as SMS or WAP, depending on the kind of device being used. These protocols are packet-based protocols that have been optimized to function over a wireless network with limited bandwidth and intermittent connectivity. These make these protocols more efficient over the existing wireless network than the standard Internet HTTP protocol.

1.4.2.2 Recognizing and Authenticating the Wireless Device

When the wireless network's base station receives the request, it requests the mobile device to identify itself in order to proceed with authentication. Once the WAP Gateway and Wireless Application Server have established a session, the WAP Gateway passes information about the specific web request to the Wireless Application Server. The message header encodes information such as the user's identity, the device the user is accessing the Internet with, the geographical location of the user, and the specific web address or service that the user is accessing. This information is used by the Wireless Application Server to personalize the interaction with the customer.

1.4.2.3 Establishing the Wireless Session

Once authentication is successful, the service provider accepts the call and establishes a connection with the mobile device. The request is sent from the base station over the wireless network using the Wireless Transport Protocol (WTP). The wireless operator's Gateway receives the request.

1.4.2.4 Translating the request over the Internet

A gateway converts the request from the cellular network protocol into the standard Internet HTTP protocol before the request is passed from the Wireless network to the traditional Internet. (The cellular network protocol is not the standard Internet protocol). For WAP- enabled devices, a WAP gateway converts WTP to HTTP. The gateway not only maps the message from one protocol to another, but also knows how to pass the message from the Wireless network to the traditional Internet infrastructure. Other gateways include Voice gateways and SMS gateways.

1.4.2.5 Connecting to the Application Server

After the Gateway converts the wireless request (which is defined by a specific phone number) to a URL for a specific web site, the message is sent as a standard Internet request to the Wireless Application Server that sits at the specific URL or web address being accessed. The Application Server and Gateway then authenticate to each other and establish a session.

1.4.2.6 Recognizing the User's Information

Once the Gateway and Wireless Application Server have established a session, the Gateway passes information about the specific Web request to the Wireless Application Server. The message header encodes such information as the user's identity, the device with which the user is accesses the Internet, the geographical location of the user, and the specific Web address or service that the user accesses. This information is used by the Wireless Application Server to personalize the interaction with the user.

1.4.2.7 Processing the Wireless Request

When the Wireless Application Server receives the content request it processes it in three steps in which the content request is adapted to the content from the wireless application being accessed, customized for the user, and transformed to the specific device being used

- Step 1: Adapting

  Content adaptation essentially involves aggregating the content from the application being accessed in an XML format. Any application, that outputs XML, will be automatically delivered to any device, over any network, with any protocol by Oracle9*i*AS Wireless.

- Step 2: Customizing the Content for Every User

  Oracle9*i*AS Wireless also recognizes the user's session context and customizes the services being rendered to the individual user. Oracle9*i*AS Wireless allows users to configure their own customized portal choosing which services they would like to see, setting up notification services, and personalizing services based on the device they are accessing the Internet from and their geographical location (Location-based Services).

- Step 3: Adapting the Content to the Appropriate Device and Network

  Finally, since each user has the ability to use one or more different devices to access the Internet and each device speaks a different markup language, Wireless transforms the content, rendering it to the markup language appropriate to the device being used.

Many wireless application servers are limited both in the range of content they can adapt and in the variety of devices to which they can render content. Typically, wireless application servers render content only to devices that speak WAP, WML and HDML. Not only can Wireless be a WAP server, but, through its usage of XML, it can translate any source content to any format for any device.

1.5 Oracle9*i*AS WirelessCore and Services

Wireless simplifies wireless development and deployment by providing the ability to deliver any content to any device, with any protocol and across any wireless network with the Oracle9*i*AS Wireless core. Oracle9*i*AS Wireless includes a set of wireless services such as PIM (and Email), Push, and Location Based Services that enhance application abilities and leverage traits. Wireless leverages open standards such as XML, Apache and J2EE, to deliver a high performance and scalable wireless infrastructure.

1.5.1 The Core

The Wireless core is the framework that gives application developers independence from the underlying networks, protocols, devices, gateways and other wireless complexities. The core normalizes the wireless complexities to one protocol and one language, HTTP and XML. Wireless is based on open J2EE, Apache, and XML standards for easy integration with existing and future technologies.

To render an application to any device, a developer needs to create any application, which outputs XML, and then point the Wireless core to the application with a URL. The core automatically eliminates the complexities associated wireless technologies.

The application can then be accessed by any device or voice, at the same time taking advantage of individual device's features.

1.5.1.1 Adapters

Wireless uses only one main protocol adapter, the HTTP Adapter, to create a mobile application from any HTTP and XML server. Wireless ships with HTTP and OC4J (J2EE) Protocol Adapters. The core, using the protocol adapters, fetches the application XML content and prepares it for device adaptation. The HTTP adapter supports the HTTP protocol and retrieves content from applications over HTTP. The OC4J (J2EE) adapter fetches content from Java Servlet and JSP based applications running within the same J2EE container (OC4J) as Wireless.

1.5.1.2 XML Application Framework

The XML application framework is based on XML and HTTP. This provides simplicity and power to application developers. Advanced HTTP/XML APIs, service linking, location awareness, and context information give developers the ability to quickly develop applications with maximum efficiency. Each application created in the XML application framework is be multi-channel to be accessed wirelessly through push, offline, and voice.

1.5.1.3 Device and Network Adaptation

Device and network adaptation automatically transform and optimize the application content to any device and network. As a result, devices that access the content retrieve optimized data. Supported devices include two-way pagers for asynchronous services (SMTP/SMS), all WAP devices, Voice access through regular phone lines, PDA devices.

1.5.1.4 Runtime APIs

The Wireless runtime uses the Oracle9*i* database as the repository for storing persistent application objects. Runtime APIs provide the functionality to manipulate the platform's persistent data objects stored in the Oracle9*i* Database repository. The Wireless APIs can customize the runtime behavior of the server. For example, the APIs can provide a different authentication scheme or a customized device identification mechanism. Wireless also provides an extension framework, which allows for plug-in of additional logic, such as logging or system monitoring that does not change the runtime behavior.

1.5.1.5 Wireless Webtools

Wireless provides a complete web-based tool to manage your wireless business. The Service Designer is used by developers to manage the applications, the Content Manager is used to manage the end user's view, the User Manager controls the users, groups and access control and the System Manager monitors the servers and performance.

1.5.1.6 Customization

Customization and personalization make applications manageable by understanding visitors' needs based on their roles and preferences. For example, customization enables information to be presented specifically to the needs of a user, whether the user is a customer, supplier, or an employee.

The advanced customization service includes alerts and data feeds. Alerts are in a publish-subscribe model and can be event-based or time-based. Event-based alerts can be based on changing events: a change in a stock price, a change in a time of a meeting, or a decrease in inventory. Time-based alerts can be based on a timed event. For example, reoccurring meetings, and appointments.

The alerts monitor and retrieve content through data feeds. Data feed content can be in multiple formats, including delimited files (CSV), HTML, or XML. The data feed can be transferred through HTTP, Local File, FTP, SQL and other applications

1.5.1.7 Push/SMS Service

Push / SMS Service provides comprehensive support for messaging. The push / SMS Service is built on a scalable message delivery architecture that can handle large volumes of messages to many different types of devices. It also provides several ways to manage and track your messages, including status of message delivered. The Push / SMS Service allows you to add your own business logic to it, to allow generating billing and routing of messages. The open architecture allows integrating into the user and device preferences of the Wireless platform. You can create distribution lists of recipients of push messages. Recipients receive messages on the device of their choice, without having to write device-specific applications.

Transport

The transport system offers a unified messaging interface to send and receive messages using any communication protocol, such as SMTP and SMS. It also features an open protocol architecture so that the system can be easily extended to support any other existing or new protocols in the future. The APIs to access the transport system are in the Java programming language.

The Push Web Service offers similar functionality to the messaging capability of the transport system. However it is set up as a SOAP-based Web Service, hence it is accessible over the network instead of requiring coding against the Java APIs that come with Wireless. By using Wireless, messaging applications can be built independently of locale relative to the Wireless installation itself. The transport system is available to anyone with an Wireless instance that the Push Web Service can access remotely. The Push/SMS Service offers a comprehensive, powerful and flexible mechanism for building messaging applications.

1.5.1.8 Transcoding

The Wireless transcoding service allows applications developed for a particular device or markup to be reformatted for other devices, including voice. Wireless supports a content adaptation service and a translator service. These services increase time to market and decrease development efforts with code reuse.

The Web Content adaptation service allows to you to quickly extend your existing legacy Web application to any wireless device. Wireless can connect any Web resource, like an HTML page, and acquire content for reformatting. The content is transformed to the Wireless XML format and then rendered to the requesting device' markup language. Web integration beans provides an abstraction and masks the complex nature of input and output elements involved in Web service transactions.

The WML translator delivers existing WML (WAP) applications to non-WML devices. The goal of the WML transcoding service is to provide a simple way for companies with existing WAP services to break the barrier of device-specific applications. The most commonly used wireless language is WML. It follows XML standards, having a Document Type Definition (DTD) that all WML documents follow. WML has different syntax and behavior from other device specific languages such as HDML. Wireless translates the WML into XML as a common language for wireless devices that hide the device-dependent complexity. The Wireless XML schema defines the basic structures that exist in WML. The structures are then rendered into any mobile device and even in voice. The translation process retains all formatting from original application.

1.5.1.9 Offline Management

Offline Management is used in cases where mobile connectivity is nonexistent or low. This gives your users the ability to use applications without any network access. When Internet connection is available again, the device user can synchronize to update the server with the new information. Oracle9*i*Lite provides this ability.

Oracle9*i*Lite is an integrated set of technologies that provide critical infrastructure for developing, deploying, and managing offline mobile applications. Oracle9*i*Lite provides necessary framework businesses need to extend the enterprise applications to all of today's popular mobile platforms: Palm OS, Symbian EPOC, Microsoft Windows CE, and Microsoft Windows 95/98/NT/2000.

1.5.1.10 Location Based Services

Location-based services greatly improve mobile applications by making them easier to use and providing quick access to timely and critical information. Companies that take advantage of location-based technologies can greatly enhance the value of their applications. Wireless location-based Service not only reduces the number of inputs and lowers the time required to obtain information, but also derives improved efficiencies, enabling access to information that is immediately relevant to users, such as maps, driving directions, traffic reports, or nearby businesses and services.

The performance and capability requirements expected for wireless location-based service can easily approach that of a top internet portal—that is, millions of queries on a daily basis, hundreds of concurrent transactions, and millisecond query-response times. When you build on Oracle9*i*, Oracle Spatial, and Wireless, you have the assurance that your location-based services solution will be scalable, reliable, and secure. In particular, it will be able to handle the unique storage and CPU-intensive processing inherent in location queries (street routing, proximity searches, and map rendering).

Wireless location services include:

Geo-coding

Automatic and Manual Mobile Positioning, Routing and Navigation

Mapping

Users can input their location or have their location automatically detected. In order to be automatically detected, Oracle9*i*AS Wireless easily integrates with vendors.

Privacy and the security of privacy-related information are important concerns in a location acquisition system. The location services provide a privacy management component that allows users to view and edit their privacy settings, to enable and disable the positioning operation on themselves, and to authorize one or more people (a mobile community) to obtain positioning information on them within certain time frames. All capabilities are accessible through public APIs.

1.5.2 Mobile PIM and Email

The Personal Information Management (PIM) Service modules are based on standard protocols, allowing a simple integration into existing environments. The Mobile Email client gives access, from any mobile devices, to any IMAP or POP3 server. This includes such servers as Microsoft Exchange and Lotus Domino. The Mobile Directory client connects to any LDAP directory server. The Mobile Calendar client integrates natively with Exchange and Lotus Servers, and through published interfaces, they enable customization to support any calendar server.

The PIM solution has a single "Universal UI", used across all back-ends. The idea is to have PIM business objects between the UI and the backend implementation, so that the same UI can be used for different backends. The same "Universal UI" can be reused or any new backends that may hit the market.

1.5.3 m-Commerce and Billing

The Wireless m-Commerce Service is a set of modules that securely stores user profiles, supplies information authorized by users for third party applications, and interfaces with on-line payment mechanisms to complete transactions. It also translates existing WML applications into Mobile-XML, and uses Formfiller to map forms and spare your customers from the frustration of typing in mobile devices.

The m-Commerce Service is automatically installed along with Oracle9*i*AS Wireless. No extra installation is necessary.

1.5.4 Mobile Studio

The Mobile Studio is an online environment for quickly building, testing and deploying wireless applications. It lets any developer, systems integrator or independent software vendor quickly develop mobile applications that are immediately accessible from all devices. This unique, next generation development environment allows companies to benefit from faster time to market, increased productivity, and a dramatically simplified testing cycle, while providing access to the latest mobile applications and tools. The Studio enables you to focus on your business logic, which is your core competency, rather than on device complexity.

The Studio's build-test-deploy model presents a hosted approach to developing dynamic content. You do not download any software or tools to start using the Studio; instead you access the Studio Web site, register, and log in. Once authenticated, you can access the reusable modules, examples, documentation, runtime information, and other resources.

You can customize the Studio by rebranding and by moving functions around to the desired positions.

1.5.5 Security

Secure wireless access to banks, enterprises, m-Commerce applications, or any other source of sensitive data is a primary concern for enterprises, carriers and application developers. However, with an ever-expanding and evolving labyrinth of wireless infrastructure (mobile devices, protocols, carriers, providers, and accompanying hardware) the problem of security simply cannot be solved in one homogeneous way. Depending on your applications, Wireless supports many techniques to satisfy your end-to-end security requirements. Wireless is built on open standards that support integration with standard security technology and third-party systems.

Oracle builds security models designed to meet the sophisticated security needs for applications such as banking, e-commerce, self-service, and CRM as well as those extending enterprise office applications to a mobile work force. Wireless utilizes such encryption technology as Wireless Transport Layer Security (WTLS), Secure Sockets Layer (SSL), Virtual Private Networks (VPN), and Public Key Infrastructure (PKI) to deliver solid end-to-end security across the Internet and the wireless network. All information, such as mWallet data and user profile data, is encrypted and stored in the secure Oracle9*i* Database.

Security-related issues may be generally classified into the following categories:

Table 1–1 Security-related Issues

Issue	Description
Privacy	Ensures that only the sender and the intended recipient can read the contents of a message (such as credit card numbers, account numbers).
Encryption and decryption	Allows two communicating parties to scramble and unscramble information they send to each other via special keys only they possess. In transit, this information is scrambled and unintelligible to any eavesdropper.
Integrity	Ensures that information is not tampered with in transit to the recipient.
Digital Signatures	Using an encrypted one-way hash algorithm, it is possible to detect at the receiving end, even if a single character has been changed. The values of the hash are unique for the hashed message, and the hash values will not expose the message since the hash is one way only.

Table 1–1 Security-related Issues

Issue	Description
Authentication	Ensures that all parties are who they claim to be such that there is no *spoofing* (no party masquerades as a legitimate entity) and misrepresentation (misleading purpose)
Digital Certificates	The process of confidently confirming the identity of one party by another party. Typically, a client communicates with a server and both client and server can be authenticated through passwords (name and password pairs) or certificates (proof of ID from an authorized source)
Non-repudiation	Ensures that a party to a genuine transaction cannot falsely deny their participation
Digital Certificates and Signatures	These are either password based or certificate based and act as proof that a designated party commissioned the transaction.

Wireless security can be illustrated by a WAP network's enforcing end-to-end security. The issues underlying WAP network security are:

Wireless Network Security: From the wireless device to the WAP gateway, a WAP 1.2 compliant network speaks the WTLS (Wireless Transport Layer Security) protocol. WTLS is a close relative of SSL and uses two kinds of certificates to manage encryption and authentication - WTLS server certificates (defined as part of WAP 1.1) are used to authenticate a WTLS server to a WTLS client and to provide a basis for establishing a key to encrypt (a handset); and WTLS client certificates (defined as part of WAP 1.2) are used to authenticate a WTLS client to a WTLS server. Both types of certificates are like standard SSL certificates except that two different certificate formats are defined - X.509 certificates (as in SSL) and WTLS mini-certificates which are functionally similar but are smaller and simpler than X.509 to facilitate processing in a resource constrained handset environment. Additionally, the mini-certificates also implement certification revocation methods that are more efficient over the wireless network than the traditional OCSP protocol.

Gateway to Wireless Application Server Security: A wireless gateway typically performs a security intermediary function such as bridging a WAP / WTLS protection environment on the wireless side with a HTTP / SSL protection environment on the wired side.

Encryption and User Authentication: When a wireless request is sent over the Wireless Network, the following steps occur:

1. The Carrier authenticates that the user is a valid wireless network user before completing the call and letting the user access the network.

2. If the user is a valid user, the call is completed and the WAP Gateway receives the WAP request. The gateway and the client then perform a standard WTLS handshake that both encrypts the communication and authenticates the gateway to the handset and vice versa.
3. The Gateway opens a HTTP session to the Oracle9*i*AS Wireless and conducts a standard SSL handshake with it - this authenticates the Gateway to the Oracle9*i*AS Wireless server and vice versa.
4. The user then accesses his or her personal portal and carries out a standard username and password based login; note that if both communication over the wireless network and between the wireless gateway and Oracle9*i*AS Wireless are secure (i.e. if the wireless network supports WTLS) then the username and password combination is not passed in the clear.
5. The user then accesses a web service. The wireless service either accepts the user's identity passed to it through the Wireless adapter as a bind variable or can ask the user to re-authenticate them again using a username and password.

In addition to network security, application security is necessary to ensure that the wireless applications protect the integrity of the user's information and the data center's information. Wireless supports application-level security with SSL and WTLS. In addition, Wireless uses a secure ACL (Access Control List) to ensure that the appropriate user is mapped to the desired information. Wireless is built on open standards that allow for easy integration with existing security systems to offer end-to-end mobile security.

Part II

Oracle9*i*AS Wireless XML Developer's Guide

Part II contains information about Oracle9*i*AS Wireless XML development.

2

XML Overview

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

- Section 2.1, "What is XML?"
- Section 2.2, "Relationship between Oracle9iAS Wireless XML and HTML"
- Section 2.3, "Why use Oracle9iAS Wireless XML?"
- Section 2.4, "How Does Oracle9iAS Wireless XML Work with Oracle9iAS Wireless?"

2.1 What is XML?

XML stands for eXtensible Markup Language. It can be best described as portable data. XML was recommended by the World Wide Web Consortium (W3C) in 1998. Since then, XML has quickly become the standard way to identify and describe data on the Web. Namespaces were added to XML in 1999. Namespaces describe a way to distinguish between two XML elements with the same name in different documents. This prevents the possibility of collision between element names among documents.

XML is a subset of SGML (Standard Generalized Markup Language), optimized for delivery over the Web. An XML document consists of a single root element. Every *start-element* must have a matching *end-element* (this property of XML documents is called *well-formedness*). Additionally, attributes of an element must be guarded by quotes.

XML documents can also be subjected to structural and global constraints, which are described by schema languages (such as document type definition [DTD] and XML Schema). An XML document is said to be valid if it satisfies the constraints described by a schema language. DTD is a weak schema language defined as part of the XML 1.0 specification and does not follow XML syntax. XML schema was

recommended by W3C in 2001. XML schema is a powerful schema language that specifies a rich set of constraints. The XML schema itself is an XML document.

2.2 Relationship between Oracle9*i*AS Wireless XML and HTML

HTML tags elements in Web pages for presentation by a browser (for example, `<bold>Oracle</bold>`).

XML tags elements as data (for example, `<company>Oracle</company>`).

You can use XML to give context to words and values in Web pages, identifying them as data instead of simple textual or numeric elements. Well formedness of XML documents makes XML processing easier and more efficient.

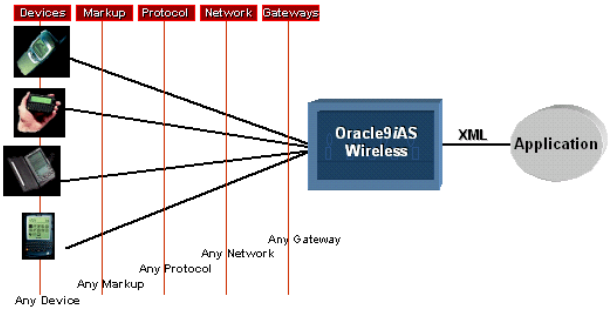
2.3 Why use Oracle9*i*AS Wireless XML?

Consider the following XML document:

```
<address>
<first-name>Chandra</first-name>
<last-name>Patni</last-name>
<street>400 Oracle Parkway</street>
<zip>94065</zip>
</address>
```

In this example, the element names self-describe the data they encapsulate. This XML document can be transformed into HTML using another XML document called an XSL stylesheet. This same XML document can be transformed into WML using another XSL stylesheet. The document can then be displayed on a WAP device. This ability of XML makes it suitable for representing and delivering portable data to various devices. XML content are also *future-proof*; another stylesheet can be used to deliver the content to any future device. Therefore, XML transformation can be done programmatically *on-the-fly*. Oracle9*i*AS Wireless provides a framework to do exactly the same thing. It allows content represented by XML format defined by an Oracle9*i*AS Wireless schema to deliver content to any device at any time.

Figure 2–1 Delivering content to different devices



2.4 How Does Oracle9*i*AS Wireless XML Work with Oracle9*i*AS Wireless?

At the core of Oracle9*i*AS Wireless, XML from an application is transformed to device-specific markup languages using XSL transformation. Oracle9*i*AS Wireless provides a framework for interacting with applications and transforming XML to device-specific markup languages. Oracle9*i*AS Wireless provides an XML schema, elements of which can be used to build user interfaces to render application content to any device.

3

Displaying and Formatting Content

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

- Section 3.1, "Hello World Example"
- Section 3.2, "Formatting the Display"
- Section 3.3, "Wireless Graphics"

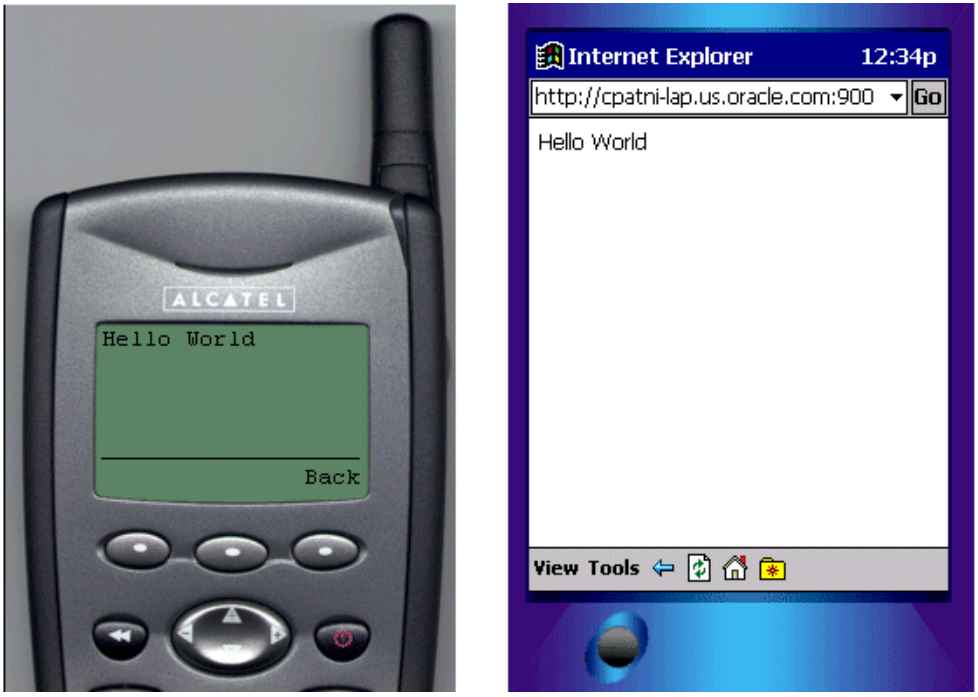
3.1 Hello World Example

The first example shows how to display the traditional "Hello World" content on a mobile device.

3.1.1 HelloWorld.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
  <SimpleText>
   <SimpleTextItem>Hello World</SimpleTextItem>
  </SimpleText>
 </SimpleContainer>
</SimpleResult>
```

Figure 3–1 Hello World content on mobile devices



In this example, XML is transformed into the device-specific markup language to render on the displays of a pocket PC and a telephone. This example demonstrates the power of XML; application programmers need not have any knowledge of the target device. Oracle9*i*AS Wireless takes care of rendering XML into the various device screens. The following section explains the XML elements, tags and attributes used in the above example. Additionally, other tags will be discussed which can be used to display and format content on device screens or voice browsers.

3.1.2 DOCTYPE Declaration

It is recommended that the XML documents authored for Oracle9*i*AS Wireless should have DOCTYPE declaration specifying the schema version. For backward compatibility (in the absence of DOCTYPE declaration), the stylesheet for Oracle9*i*AS Wireless Edition 1.0 will be applied. However, if 1.0 stylesheets are not available to Oracle9*i*AS Wireless runtime, then Oracle9*i*AS Wireless 1.*x* stylesheets

will be used regardless of DOCTYPE declaration. If no 1.*x* stylesheets are not found, an error will result.

3.1.3 SimpleResult

SimpleResult is the root element of the Oracle9*i*AS Wireless XML schema. Every valid Oracle9*i*AS Wireless XML document must have *SimpleResult* as its root element. SimpleResult can contain multiple *SimpleContainer* blocks to allow for multi-card decks.

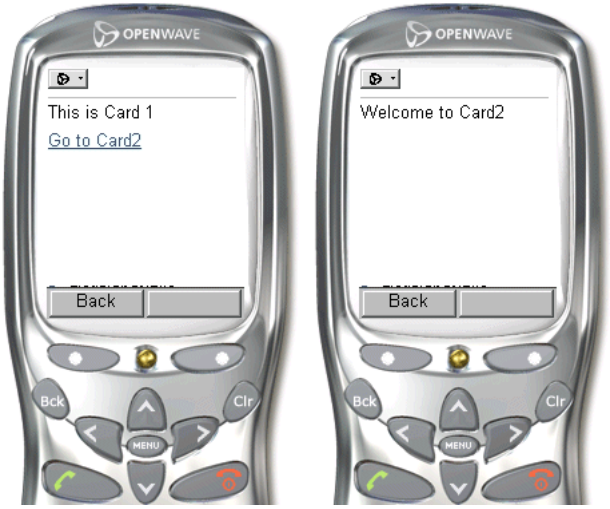
3.1.3.1 SimpleContainer

SimpleContainer is the root of all major block constructs such as Form, Menu and Text. Elements such as menu, text and form items can act as cards in the deck. DeckExample.xml demonstrates the usage of SimpleText as a placeholder for cards. Considering the limitation of target devices and deck size restrictions on devices, judgment should be exercised in the number of cards per deck and the total content size in a single request.

3.1.3.2 DeckExample.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
  <SimpleText id="card1">
   <SimpleTextItem>This is Card 1
    <SimpleBreak/>
    <SimpleHref target="#card2">Go to Card2</SimpleHref>
   </SimpleTextItem>
  </SimpleText>
  <SimpleText id="card2">
   <SimpleTextItem>Welcome to Card2</SimpleTextItem>
  </SimpleText>
 </SimpleContainer>
</SimpleResult>
```

Figure 3–2 Cards displayed on mobile telephones



3.1.3.3 SimpleText, SimpleTextItem

Content of SimpleTextItem are usually translated into paragraphs. SimpleTextItem can be grouped using SimpleText element. SimpleText element contains one or more SimpleTextItem. The id attribute of SimpleText tag can be used to refer to SimpleText elements as a deck. SimpleText is rendered on a separate card on WML and HDML devices. SimpleHref can be used as a child of SimpleTextItem similar to HTML anchor. See Section 4.3.1, "SimpleHref, SimpleTimer" for more information on SimpleHref. The *deviceclass* attribute of SimpleText and SimpleTextItem take values “pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", and "messenger" which directs processing for either small screen clients or voice clients. In the absence of the deviceclass attribute, the content will be rendered to both small screen devices and voice enabled devices. By default, text-to-speech (TTS) synthesis is used to represent the text enclosed in these tags. SimpleAudio tag in conjunction with deviceclass attribute can be specified to override the default behavior. For a better user experience, do not use TTS whenever voice feed is available. For voice interfaces SimpleAudio may be used. Refer to the following snippet of code for usage.

```
<SimpleText>
 <SimpleTextItem>
  <SimpleAudio src="http://www.domain.com/filename.wav" deviceclass="voice">Alt
text for TTS if the wave file is not found.
 </SimpleAudio>
```

```
 </SimpleTextItem>
 <SimpleTextItem deviceclass="microbrowser"> Text for small screen devices
 </SimpleTextItem>
</SimpleText>
```

Note: The .wav file specified must be in CCITT mu-law, 8 bit, 8kHz.

3.2 Formatting the Display

3.2.1 SimpleBreak, SimpleStrong and SimpleEm

These elements are used for fine-tuning the display of text content on a screen. SimpleStrong displays enclosed text in a stronger representation, usually bold. SimpleEm displays the enclosed text with emphasis, usually displayed as italicized text. For voice-enabled applications, level attribute can be used to specify the level of emphasis. Permissible values for level attribute are: strong, moderate, none and reduced.

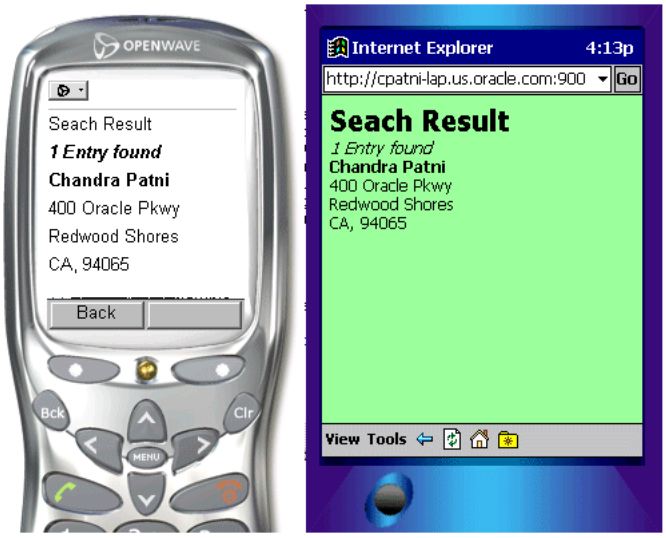
SimpleBreak creates a new line on the page on which the tag is placed. The rule attribute can be used to display a line <hr>, for HTML output. Deviceclass can be used for directive processing of small screen or voice enabled devices, or both. For voice-enabled applications, SimpleBreak enables you to specify msecs and size attributes to control the break while delivering text. See the following example for details.

3.2.1.1 FormattingExample.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult bgcolor="99ff99">
 <SimpleContainer>
  <SimpleText>
   <SimpleTitle>Seach Result</SimpleTitle>
   <SimpleTextItem>
    <SimpleEm level="strong">1 Entry found</SimpleEm>
    <SimpleBreak msecs="500"/>
    <SimpleStrong level="strong">Chandra Patni</SimpleStrong>
    <SimpleBreak/>400 Oracle Pkwy
    <SimpleBreak/>Redwood Shores
    <SimpleBreak/>CA, 94065
```

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

Figure 3–3 Results of formatting example



3.2.2 Tables and Basic Formatting Example

3.2.2.1 SimpleTable, SimpleTableHeader, SimpleTableBody, SimpleRow and SimpleCol

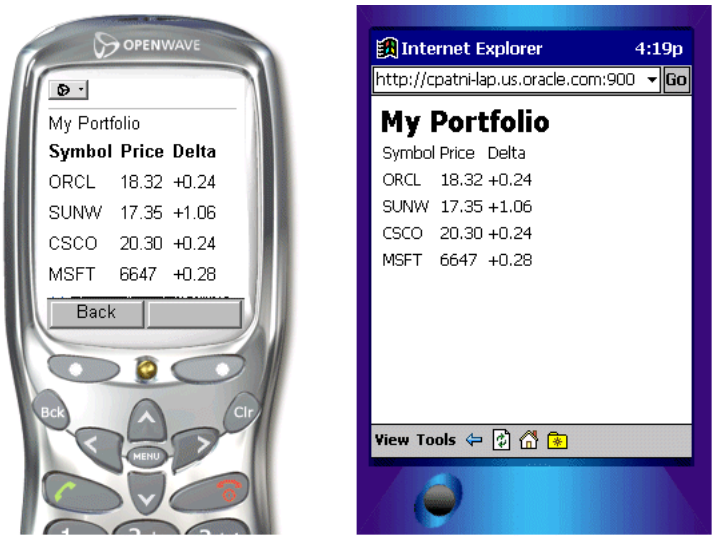
SimpleTable displays a table. A table consists of a header and body which are abstracted by SimpleTableHeader and SimpleTableBody, respectively. The body of a table consists of SimpleRow and SimpleCol elements. Images can be used in tables cells. TableExample.xml provides an example of the table elements.

3.2.2.2 TableExample.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
```

```
  <SimpleTable >
   <SimpleTitle> My Portfolio </SimpleTitle>
   <SimpleTableHeader>
    <SimpleCol>Symbol</SimpleCol>
    <SimpleCol>Price</SimpleCol>
    <SimpleCol>Delta</SimpleCol>
   </SimpleTableHeader>
   <SimpleTableBody>
    <SimpleRow>
     <SimpleCol>ORCL</SimpleCol>
     <SimpleCol>18.32</SimpleCol>
     <SimpleCol>+0.24</SimpleCol>
    </SimpleRow>
    <SimpleRow>
     <SimpleCol>SUNW</SimpleCol>
     <SimpleCol>17.35</SimpleCol>
     <SimpleCol>+1.06</SimpleCol>
    </SimpleRow>
    <SimpleRow>
     <SimpleCol>CSCO</SimpleCol>
     <SimpleCol>20.30</SimpleCol>
     <SimpleCol>+0.24</SimpleCol>
    </SimpleRow>
    <SimpleRow>
     <SimpleCol>MSFT</SimpleCol>
     <SimpleCol>6647</SimpleCol>
     <SimpleCol>+0.28</SimpleCol>
    </SimpleRow>
   </SimpleTableBody>
  </SimpleTable>
 </SimpleContainer>
</SimpleResult>
```

Figure 3–4 Results of tables and basic formatting example



3.3 Wireless Graphics

3.3.1 SimpleImage

This element is used for displaying a WBMP or BMP graphic on small screen devices. GIF is also supported for HTML clients. The image resolution supported is 2-bits. *src* is a compulsory attribute of the SimpleImage element. Unlike HTML, the extension of the image is not specified for Oracle9*i*AS Wireless. Appropriate extension will be appended for the target mark up language. All the images with appropriate extension (.wbmp, .bmp) should be provided in the target directory. See the following example for usage.

Devices do not support a single format of an image. As of Release 2.0, Oracle9*i*AS Wireless does not support dynamic image extension conversion. Application developers can suggest the available formats of the image by specifying *available* attribute. The *available* attribute is the list of whitespace-separated values of **jpg, gif, g2.gif, bmp and wbmp** formats. g2.gif is grayscale / depth 2 image format, typically used for Palm. Transformers apply the following rules to determine the format.

The transformer checks the *available* format with the list of supported Images formats provided by the server. The server has a preferred Image formats property

for each logical device. This list can contain one or all of the formats supported by the available attribute.

- if there is a match the image is rendered
- if there is no match and *alt* attribute exists, *alt* text is rendered
- else the image is ignored

3.3.2 ImageDisplay.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
  <SimpleText>
   <SimpleImage src="http://portal.oraclemobile.com/other/oow/oramobile"
alt="Welcome To OracleMobile"/>
  </SimpleText>
 </SimpleContainer>
</SimpleResult>
```

Figure 3–5 Results of image example

3.4 Enhancing with Audio for Voice Access

3.4.1 SimpleAudio and SimpleSpeech

The SimpleAudio element can be used for playing audio. The file specified by the *src* attribute must be in 8-bit mulaw format. The SimpleSpeech element may be used to control prosody pitch and other VoiceXML text-to-speech engine parameters. For example, the class attribute can be used to specify the *sayas* text-to-speech output as phone, date, digits, literal, currency, number or time. See the following example for usage.

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
  <SimpleText>
   <SimpleTextItem>
    <SimpleAudio src="welcome1.wav">Welcome to Oracle Mobile, India Development
Center</SimpleAudio>
    <SimpleBreak/>
    <SimpleAudio src="welcome2.wav">You can contact us at phone number
</SimpleAudio>
    <SimpleBreak/>
    <SimpleSpeech class="phone">
     <SimpleAudio src="phone.wav">91 080 552 8335</SimpleAudio>
    </SimpleSpeech>
   </SimpleTextItem>
  </SimpleText>
 </SimpleContainer>
</SimpleResult>
```

Figure 3–6 Results of SimpleAudio and SimpleSpeech example

3.4.2 Recommendation for Voice Navigation

While writing applications for Oracle9*i*AS Wireless, developers should consider voice navigation at design time. Well-designed voice applications tend to have different semantics than small screen devices and desktop applications. Although Oracle9*i*AS Wireless automatically provides an audio interface for service, the system is not intended to be a *speech-controlled small-screen device browser*, where speech is added as an afterthought. Application developers should develop services that have appropriate small-screen and speech interfaces in their own right, and the respective strengths of these different devices can be used to advantage.

The development path for beginners should follow this model:

1. Write a basic version of the service using exactly the same flow and markup for small-screen devices and audio interfaces.
2. Test on small-screen devices and voice telephones. If it is acceptable, you are done.

For a large class of services, particularly menu-driven services that provide information, the method works surprisingly well. If one or another interface seems clumsy, there are several things that can be done to improve it.

1. First, there are a number of attribute values that can be adjusted to enhance the interface for one of the device classes.
2. Second, if that is insufficient, one can selectively include or exclude certain elements from the user interface depending on the *deviceclass*.
3. Finally, one can alter the user interface flow by selectively following different paths through a service, again, depending on the *deviceclass*.

4

Application Navigation

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

- Section 4.1, "Introduction"
- Section 4.2, "Basic Navigation"
- Section 4.3, "Document Linking"

4.1 Introduction

Before examining the properties of writing mobile XML to handle text formatting from a small device and voice perspective, this chapter will help you gain the skills to write effective user interfaces to capture the required business logic with the least amount of effort by mobile users. We will examine the details of creating Oracle9*i*AS Wireless XML pages containing navigation elements such as menus, hyperlinks, email, help, and cover forms. The elements necessary to build a form are different from a menu as these will be the core elements needed for a wireless developer to build an effective mobile application that simplifies user input without compromising a rich feature set across different devices.

Because voice navigation is inherently more complicated than in small screen devices, this chapter focuses on the fundamentals of Oracle9*i*AS Wireless XML for small devices and highlights the required voice additions.

Menus allow consumers of services to simply navigate to a predefined choice and enable different URLs to be invoked for a given choice. Forms, on the other hand, typically differ from Menus in that there is one target which dictates the user's next page based on user input.

4.2 Basic Navigation

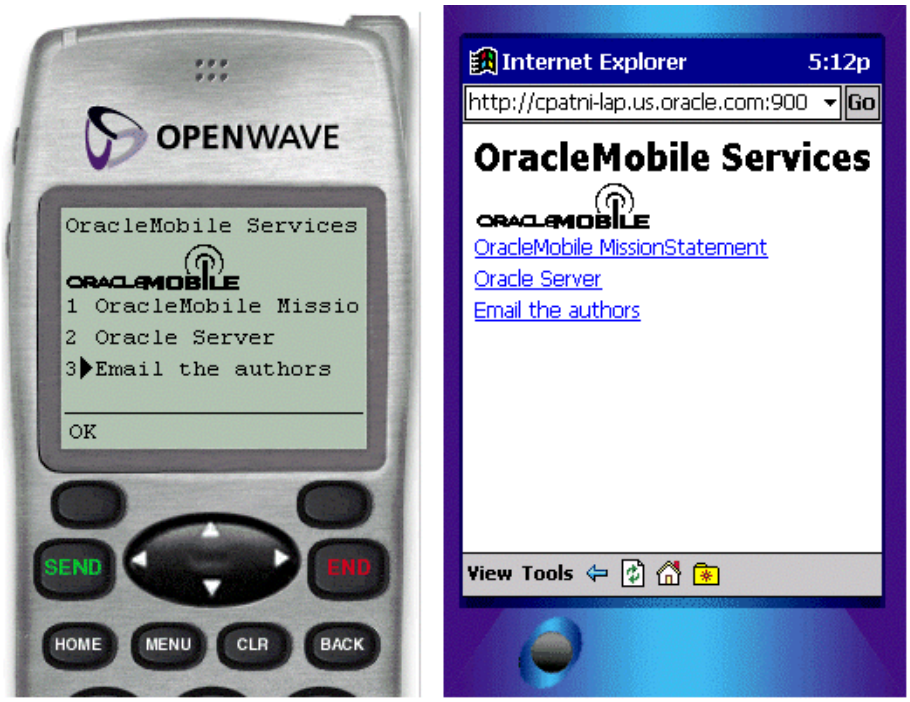
4.2.1 SimpleMenu, SimpleMenuItem

The SimpleMenu element represents a single menu with selectable menu items defined by SimpleMenuItem elements. It is possible to add Images to the top of each Menu. However, one needs to avoid using large titles and images. See ***SimpleMenuExample.xml*** for an example.

4.2.1.1 SimpleMenuExample.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC " = //ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
  <SimpleMenu>
   <SimpleTitle>OracleMobile Services
    <SimpleImage src="
http://portal.oraclemobile.com/other/oow/oramobile"alt="Oracle Software
Powers the Internet"/>
   </SimpleTitle>
   <SimpleMenuItem target="mission.xml">OracleMobile
MissionStatement</SimpleMenuItem>
   <SimpleMenuItem target="timer.xml">Oracle Server</SimpleMenuItem>
   <SimpleMenuItem target="email.xml">Email the authors</SimpleMenuItem>
   </SimpleMenu>
  </SimpleContainer>
 </SimpleResult>
```

Figure 4–1 Results of simple navigation example



4.2.2 Navigating by Voice

The system reads the items of menu elements and concurrently listens for the values of the *SimpleMenuItem* element. If one of these values is recognized, then the target URL is fetched. If the user says nothing, the system will prompt the user with a system default noinput message. If the user says something and the system is unable to recognize it, the system default *nomatch* message is played. However, application programmer may control such messages. Such fail-over logic is critical for making robust voice applications. Application developers should make extensive use of such features. For menus with large number of items, voice interfaces should not read out the entire menu items to the user by setting the *autoprompt* attribute to *false*. Instead, applications should wait for user input and should only present an options list as help if requested by user. See **EnhancedSimpleMenuExample.xml** for example. Some of the tags and elements used in the application are covered later in this chapter.

4.2.2.1 EnhancedSimpleMenuExample.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC " = //ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
 <SimpleResult>
  <SimpleContainer>
   <SimpleMenu deviceclass="microbrowser pdabrowser pcbrowser micromessager
messanger">
   <SimpleTitle>OracleMobile Services
   <SimpleImage src="http://portal.oraclemobile.com/other/oow/oramobile"
alt="Oracle Software Powers the Internet"/></SimpleTitle>
   <SimpleTitle>OracleMobile Services</SimpleTitle>
   <SimpleMenuItem target="mission.xml">OracleMobile
MissionStatement</SimpleMenuItem>
   <SimpleMenuItem target="timer.xml">Oracle Server</SimpleMenuItem>
   <SimpleMenuItem target="email.xml">Email the authors</SimpleMenuItem>
  </SimpleMenu>
  <SimpleMenu deviceclass="voice" autoprompt="false">
   <SimpleTitle>
    <SimpleAudio src="title.wav">oracle mobile services
    </SimpleAudio>
   </SimpleTitle>
   <SimpleMenuItem target="mission.xml">OracleMobile MissionStatement
    <SimpleGrammer>mission statement{}| oracle mission statement{}
    </SimpleGrammer>
   </SimpleMenuItem>
   <SimpleMenuItem target="timer.xml">Oracle Server
    <SimpleGrammer>oracle server{}| server{}
    </SimpleGrammer>
   </SimpleMenuItem>
   <SimpleMenuItem target="email.xml">Email the authors
    <SimpleGrammer>email the authors{}| email{} | email authors{}
    </SimpleGrammer>
   </SimpleMenuItem>
   <SimpleCatch type="nospeech">
    <SimpleAudio src="menuOptions.wav">Please speak up. You may also say help.
    </SimpleAudio>
   </SimpleCatch>
   <SimpleCatch type="nomatch">
    <SimpleAudio src="nomatch.wav">I'm sorry, I did not understand you. Please
say that again or say help.</SimpleAudio>
   </SimpleCatch type="help">
    <SimpleAudio src="menuHelp.wav"> Help. Oracle Mobile. You may say mission
statement, oracle server or email the authors.
    </SimpleAudio>
```

```
  </SimpleMenu>
 </SimpleContainer>
</SimpleResult>
```

The output of this application on small screen devices is the same as shown above, while a typical voice session may be as follows:

```
System: oracle mobile services
User: help
System: Help. Oracle Mobile. You may say mission statement, oracle server or
email the authors.
User: I am going to trick you.
System: I'm sorry, I did not understand you. Please say that again or say help.
User: email authors
...
```

Generally, voice gateways provide a text-to-speech (TTS) engine that reads out SimpleTitle, SimpleTextItem, SimpleMenu options, SimpleFormOptions etc. For the TTS to sound intelligible, proper spacing and punctuation is required. SimpleFormOption or SimpleMenuItem should never have text punctuation unless the deviceclass has been set to a value other than “voice”. This is because the text in these tags is used to produce *speech recognition grammars*, and most grammars are foiled by non-alphabetic characters. If a developer wishes to avoid using the synthesized message, then he may specify a prerecorded audio file to be played. The location of the audio file can be specified through the <SimpleAudio> tag. End user experience of TTS is often considered unpleasant, So as much as possible, prerecorded human sounds should be used instead of TTS.

4.3 Document Linking

4.3.1 SimpleHref, SimpleTimer

For linking documents, SimpleHref can be used as a hyperlink. It can also be used to send email using the `mailto:` handler as shown in the ContactAuthors.xml and PhoneCallDemo.xml examples. Similarly, the `callto:` handler can be used for devices that are capable of making phone calls. Application developers should specify deviceclass attributes which support the call or mail feature.

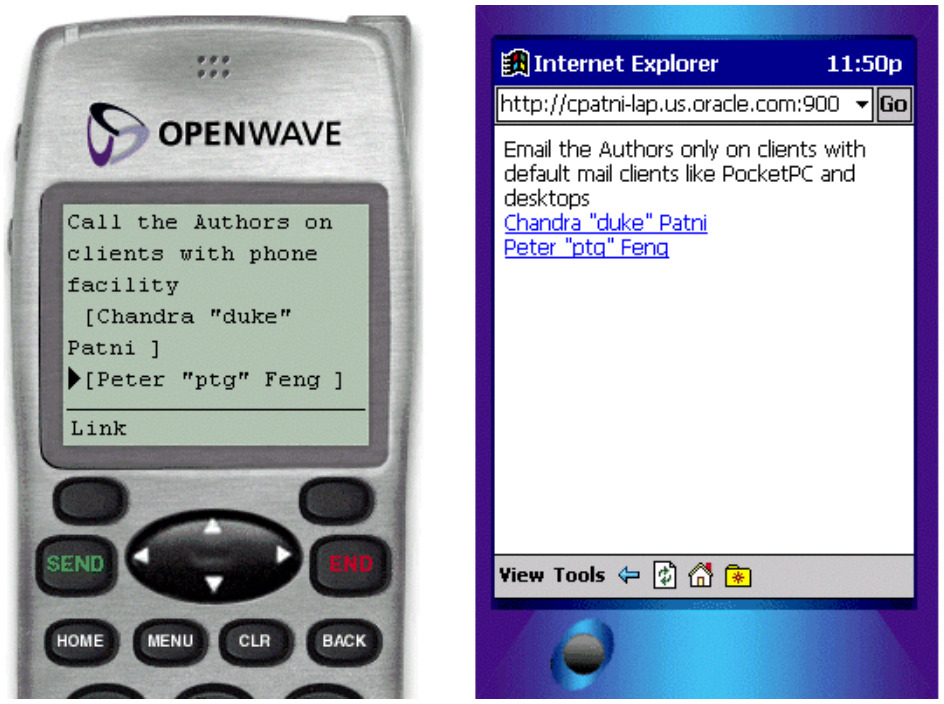
> **Note:** *static_target* attribute should be used instead of *target* whenever *callto:* or *mailto:* handlers are used which signals Oracle9*i*AS Wireless runtime not to rewrite the URLs.

SimpleTimer can be used to invoke a goto target task after a specified delay. It can be used for navigation to display a showcase promotion, sponsor information, or system-wide critical messages.

4.3.1.1 ContactAuthors.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
  <SimpleText>
   <SimpleTextItem deviceclass="pdabrowser pcbrowser  micromessenger
messenger">Email the Authors only on clients with default mail clients like
PocketPC and desktops
    <SimpleBreak/>
    <SimpleHref staic_target="mailto:chandra.patni@oracle.com">Chandra "duke"
Patni
    </SimpleHref>
    <SimpleBreak/>
    <SimpleHref static_target="mailto:peter.feng@oracle.com">Peter "ptg" Feng
    </SimpleHref>
   </SimpleTextItem>
   <SimpleTextItem deviceclass="voice microbrowser">Call the Authors on clients
with phone facility
    <SimpleBreak/>
    <SimpleHref static_target="callto:1234567890">Chandra "duke" Patni
    </SimpleHref>
    <SimpleBreak/>
    <SimpleHref static_target="callto:1234567890">Peter "ptg" Feng
    </SimpleHref>
   </SimpleTextItem>
  </SimpleText>
 </SimpleContainer>
</SimpleResult>
```

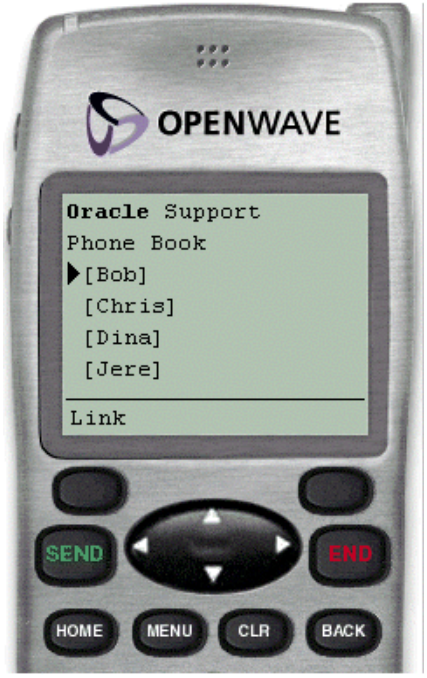
Figure 4–2 Results of the Email demo example



4.3.1.2 PhoneCallDemo.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
  <SimpleText deviceclass="microbrowser voice">
    <SimpleTextItem><SimpleEm>Oracle</SimpleEm> Support </SimpleTextItem>
    <SimpleTextItem>Phone Book<SimpleBreak/>
    <SimpleHref static_target="callto:14155551212">Bob</SimpleHref>
    <SimpleHref static_target="callto:16505551212">Chris</SimpleHref>
    <SimpleHref static_target="callto:14085551212">Dina</SimpleHref>
    <SimpleHref static_target="callto:17075551212">Jere</SimpleHref>
    </SimpleTextItem>
  </SimpleText>
 </SimpleContainer>
</SimpleResult>
```

Figure 4–3 Results of the Phone Call Demo example



4.3.1.3 SimpleAction

SimpleAction provides the ability to define a submit action, that navigates users to a new context. Mobile devices can associate a submit action to a number of input methods of the device, such as pressing a key on a WAP device or speaking a command on a voice-enabled device. SimpleAction can also be used for navigation to different pages and different cards within a deck, and overriding default behavior on voice browsers. For mobile phones, the main usage would be to override the buttons (left and right) on a wireless phone and PDAs to provide a similar navigation functionality as SimpleHref.

Like many programming languages, SimpleAction, for a given type, conforms to scoping rules. For example, if SimpleAction is defined as a child of SimpleMenu and also as a child of the enclosing SimpleContainer for a given *type*, the SimpleAction tag within the SimpleMenu overrides the SimpleAction of the SimpleContainer. If the value for *type* attribute is different, then the two SimpleActions will be active within the context. The behavior of SimpleAction is

unspecified if two elements are defined with the same *type* and same *deviceclass* values in the same context. See the following example for usage.

4.3.1.4 SimpleCache

SimpleCache enables you to specify caching policy of content either by the WAP gateway, by client browser, or both.

- Caching policy is said to be public if the WAP gateway is allowed to cache the content of a URL.
- Caching policy is said to be private if the content is only allowed to cache by the device.

SimpleCache can be specified as the child of SimpleHref, SimpleGo, SimpleMenuItem, SimpleAction etc. SimpleCache also allows users to specify the prefetch policy (if supported by browser), where a URL must be prefetched while still showing the current content. However, if the SimpleAction specifies a submit task, then caching policies are not applicable. Time to live for the cached data is specified by the `ttl` attribute, which takes milliseconds as an argument.

SimpleCache should be used when the data is sensitive or becomes stale after a specified amount of time.

4.3.1.5 SimpleMeta

SimpleMeta allows applications to specify meta information via the device browser, and pass that information to the transformers.

4.3.2 Enhancing with Voice

4.3.2.1 SimpleDTMF

SimpleDTMF specifies a VoiceXML DTMF grammar. In the voice application example the user may select menu item ‘test1’ either by saying ‘test1’ or by selecting ‘3’ on the device.

4.3.2.2 SimpleDTMF.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult public="true">
 <SimpleCache ttl="0"/>
 <SimpleContainer>
```

```
  <SimpleMenu markable="true" wrapmode="nowrap" autoprompt="false" dtmf="true">
   <SimpleTitle>Voice demo</SimpleTitle>
   <SimpleMenuItem target="deposit.jsp">Deposit
    <SimpleAction task="go" method="get"/>
    <SimpleGrammar> deposit{} </SimpleGrammar>
    <SimpleDTMF>1</SimpleDTMF>
   </SimpleMenuItem>
   <SimpleMenuItem target="HelloWorld.jsp">Withdraw
    <SimpleAction task="go" method="get"/>
    <SimpleGrammar>withdraw{}</SimpleGrammar>
    <SimpleDTMF>2</SimpleDTMF>
   </SimpleMenuItem>
   <SimpleCatch type="cancel">
    <SimpleAction target="cancel.jsp"/>
   </SimpleCatch>
   <SimpleCatch type="help">
    <SimpleAudio src="help.wav">Help. For deposit, you may say deposit or press
1. For withdraw, you
    may say withdraw or press 2.</SimpleAudio>
   </SimpleCatch>
   <SimpleCatch type="help" count="2">
    <SimpleAudio src="help.wav">Help. For deposit, you may say deposit or press
1. For withdraw, you
    may say withdraw or press 2. You may also say cancel to return to account
menu.</SimpleAudio>
   </SimpleCatch>
  </SimpleMenu>
 </SimpleContainer>
</SimpleResult>
```

4.3.2.3 SimpleCatch

SimpleCatch catches an event; it is a voice-only tag. This can be used to capture predefined voice events or error conditions such as "noinput", “nomatch”, "exit", “cancel”, “error”, “help”, “telephone.disconnect”, etc. and perform actions on them. For example on a "noinput" event the user can be given some help instructions and be reprompted for their input. The event types are specified by type attribute which is mandatory for SimpleCatch. Also, count attribute may be used for occurrences of the event. The default value is 1. It allows handling of multiple occurrences of an event in multiple ways. For example the n^{th} occurrence of an event can be handled in a different manner than the previous occurrence. In a frequently occurring scenario, it may be used for increasing details of help as count increases. See SimpleDTMF.xml for usage.

4.3.2.4 SimpleGrammar

The SimpleGrammar tag provides a customized speech recognition grammar. Using this grammar, developers can not only provide the vocabulary to listen for, but also the mapping from, utterances to data values. If the rules for such mappings are in a remote location, then the *src* attribute may be used to specify the name of the file. The following example illustrates the use of SimpleGrammar.

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
  <SimpleMenu deviceclass="voice">
   <SimpleTitle src="title.wav">Please select a freeway</SimpleTitle>
   <SimpleMenuItem target="./traffic.jsp?index=5">I 5
    <SimpleGrammar>i five{} | interstate five{} | five{} | route five{} | san
diego{}
    </SimpleGrammar>
   </SimpleMenuItem>
   <SimpleMenuItem target="./traffic.jsp?index=8 ">I 8
    <SimpleGrammar>i eight{} | interstate eight{} | eight{} | route eight{} |
alvarado freeway{} |
    mission valley freeway{} | ocean beach freeway{}
    </SimpleGrammar>
   </SimpleMenuItem>
   <SimpleMenuItem target="./traffic.jsp?index=15 ">I 15
    <SimpleGrammar>i fifteen{} | fifteen{} |interstate fifteen{} |  escondido
freeway{}| escondido{}
    </SimpleGrammar>
   </SimpleMenuItem>
   <SimpleMenuItem target="./traffic.jsp?index=805 ">I 805
    <SimpleGrammar>i eight zero five{} | i eight hundred five{} | eight zero
five{} | eight hundred five{} |
    interstate eight zero five{} | interstate eight hundred five{} | route eight
zero five{} |
    route eight hundred five{}
    </SimpleGrammar>
   </SimpleMenuItem>
  </SimpleMenu>
 </SimpleContainer>
</SimpleResult>
```

In the above example, even though the last menu option is “i eight hundred five”,

the user may say any one of the commands as specified by a '|' separated list. SimpleGrammer is a very useful construct for building user-friendly and smart voice applications. It also allows application developers to incorporate some of their localization issues. For example, "sure", "ok", "yes", "please" and "yes please" all are used to refer to "yes" (in America region) in different parts of world. Such speech diversity can be incorporated into an application using SimpleGrammer.

Note: Only lowercase ASCII characters are allowed in SimpleGrammer.

4.3.2.5 DocumentLinkingDemo.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer id=″message″>
  <SimpleTimer target=″#employeePortal″ timer=″30″/>
  <SimpleText>
   <SimpleTextItem> There will be ice cream bars in every lobby at Headquarters
to promote the use of the new employee wireless portal.
   </SimpleTextItem>
  </SimpleText>
 </SimpleContainer>
 <SimpleContainer>
  <SimpleText id=″employeePortal″>
   <SimpleImage valign=″top″ src=
http://portal.oraclemobile.com/other/oow/oraclemobile alt=″oraclemobile icon″/>
   <SimpleTextItem>Welcome to <SimpleEm>OracleMobile</SimpleEm> Employee Portal
<SimpleBreak/>
   </SimpleTextItem>
   <SimpleAction type=″SOFT1″ label=″Support″ target=″phone.xml″/>
   <SimpleHref label=″PORTAL″ id=″portal″ name=″ToPortal″ target=″form.xml″>
enterPortal
   </SimpleHref>
  </SimpleText>
 </SimpleContainer>
</SimpleResult>
```

Figure 4–4 Results of the Document Linking demo example

4.3.2.6 Mobile XML Voice Navigation Elements

Basic Voice Commands

The following basic commands are available to users at all times. The response of the system to *help* and *cancel* will generally need to be tailored to each individual service.

Main menu: Can be uttered at any time, and by default takes the user to the Oracle9*i*AS Wireless main menu.
Goodbye: To end the session with one Oracle9*i*AS Wireless instance or user may just hang up the telephone.
Exit: Same as Goodbye.
Help: For context-sensitive help>
Cancel: For aborting or restarting a dialog, as when the system has misrecognized a command or input.

4.3.2.7 Help

Help is used by voice applications to provide context-sensitive help when users invoke “help” commands. Voice interfaces should make use of Help as much possible. Unlike small screen application help, voice help is vital to the navigation of voice interfaces and therefore should be incorporated at development time. See EnhancedSimpleMenuExample.xml for usage.

5

Filling Out Forms for Data Entry and Navigation

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

5.1 Introduction

Forms provide the basic building blocks for user interactions. Forms for phones and PDAs are fairly similar, except in form factor. Like HTML forms, forms in mobile devices are used for passing name-value parameters to the server. Multiple form items can be laid out on the device screen, if supported. Therefore, a user may populate a form item in an arbitrary order. Certain format restrictions can be specified on a form item to ensure the type safety and validity of form fields. For example, it is possible to specify a restriction of five digits for US postal codes. However, most of the validation should occur on the server side. This constraint is due to the limited resources on the devices. On a voice browser, every thing must be processed by the voice gateway, which enables rich validation and exception handling at the markup language level.

5.2 Basic User Interaction

5.2.1 SimpleForm

SimpleForm is similar to HTML form, which provides an arbitrary collection of SimpleFormItem and SimpleFormSelect as a single entity. SimpleFormSelect may be used to display list, radio buttons or checkbox controls. Form has SimpleTitle as its child, and if specified, will appear as the Title of the form. SimpleForm along with SimpleBind can trigger form processing in several ways; multiple tasks can be executed upon form submission.

5.2.2 SimpleFormItem

SimpleFormItem is the equivalent of a text field, text area, password field and hidden field for desktop browsers. The type of item may be specified using the display mode attribute. It may take text field, text area, noecho or hidden. SimpleFormItem can be used to obtain input from a user. This element presents a prompt, and waits for input from the user. The content of this element, which is in parsable character format, specifies default values for the form item. For example, a login screen and guest book screen may appear as in the following example.

5.2.2.1 FormExample.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
  <SimpleForm target="login.jsp" method="post">
   <SimpleFormItem name="userName">User Name:</SimpleFormItem>
   <SimpleFormItem name="password"
displaymode="noecho">Password:</SimpleFormItem>
  </SimpleForm>
 </SimpleContainer>
</SimpleResult>
```

Figure 5–1 Results of FormExample.xml example

5.2.2.2 GuestBook.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
  <SimpleForm target="sendMail.jsp" method="post">
   <SimpleTitle>Thanks for signing my guestbook.</SimpleTitle>
   <SimpleFormItem name="Name">Name:</SimpleFormItem>
   <SimpleFormItem name="message"
displaymode="textarea">Message:</SimpleFormItem>
  </SimpleForm>
 </SimpleContainer>
</SimpleResult>
```

Figure 5–2 Results of GuestBook.xml example

5.3 Complete User Forms

5.3.1 SimpleFormSelect, SimpleFormOption, and SimpleOptGroup

These elements display a selected option list. It can display drop down list, checkbox and radio button, using the *display mode* attribute. Checkboxes or option lists may allow single selection or multiple selections using the *multiple* attribute. The items to be displayed are abstracted by the *SimpleFormOption* element. SimpleOptGroup groups SimpleFormOption elements into a hierarchy. It is useful for small screen devices, where long list of options cannot be esthetically presented in the user interfaces. The content of SimpleFormOption element is parsable character data, which specifies default values for the form item. See the following example for usage.

5.3.2 Profile.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
```

```
  <SimpleForm name="employeeinfo" target="process.jsp">
   <SimpleTitle>Your Profile</SimpleTitle>
   <SimpleFormItem name="homepage" default="http://">Homepage</SimpleFormItem>
   <SimpleFormSelect name="skills" displaymode="checkbox" multiple="true">
    <SimpleTitle>Skills</SimpleTitle>
    <SimpleFormOption value="java">Java</SimpleFormOption>
    <SimpleFormOption value="xml">XML</SimpleFormOption>
    <SimpleFormOption value="sql">SQL</SimpleFormOption>
   </SimpleFormSelect>
   <SimpleFormSelect name="nerd" displaymode="checkbox">
    <SimpleTitle>Addicted to Java?</SimpleTitle>
    <SimpleFormOption value="yes">Yes</SimpleFormOption>
    <SimpleFormOption value="no">No</SimpleFormOption>
   </SimpleFormSelect>
   <SimpleFormSelect name="location" displaymode="list">
    <SimpleTitle>Location</SimpleTitle>
    <SimpleFormOption value="Redwood Shores_CA">HQ Redwood
Shores,CA</SimpleFormOption>
    <SimpleFormOption value="Nashua_NH">NEDC Nashua, NH</SimpleFormOption>
    <SimpleFormOption value="SanFrancisco_CA">SanFrancisco,
CA</SimpleFormOption>
    <SimpleFormOption value="NewYork,NY">NewYork, NY</SimpleFormOption>
   </SimpleFormSelect>
   </SimpleForm>
  </SimpleContainer>
</SimpleResult>
```

Figure 5–3 Results of Profile.xml example

5.4 Enhancing Voice

5.4.1 SimpleGrammer, SimpleValue and SimpleDTMF

SimpleGrammar— The SimpleGrammar tag provides a customized speech recognition grammar. For further details on the use of SimpleGrammar see Section 4.3.2.4, "SimpleGrammar".

SimpleValue—The SimpleValue tag is a placeholder for dynamic information that is not known until runtime. This element is valuable for processing multiple cards within one deck and capturing client-side data validation.

SimpleDTMF—This is a keyboard binding number that is used to process input. In the example below, the formItem ZipInput would pass only 232 to the target and nothing else unless there was an error or long pause.

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
  <SimpleForm name="Starting" target="test2a.jsp">
   <SimpleFormItem name="addrInput">
```

```
    simple grammar test, please say oracle or san mateo
   <SimpleGrammar>
    oracle {bridge}|san mateo{foster city}
   </SimpleGrammar>
   </SimpleFormItem>
   <SimpleFormItem name="zipInput">
    <SimpleDTMF> 95 {232} </SimpleDTMF>
     Simple DTMF test, please press 95
   </SimpleFormItem>
  </SimpleForm>
 </SimpleContainer>
</SimpleResult>
```

5.4.2 Recommendation for Voice Forms

So far we have written the form for the small screen devices which are similar to the following form.<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>

```
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
  <SimpleForm target="guess.jsp">
   <SimpleFormItem name="guess">
    <SimpleTitle>
    I am thinking of a number between 1 and 100.
    What is your first guess?
    </SimpleTitle>
   </SimpleFormItem>
  </SimpleForm>
 </SimpleContainer>
</SimpleResult>
```

This example would work well for a small screen device. However, this is not sufficient for spoken input. Speech recognition works only when there is a very narrowly prescribed vocabulary to listen for. Descriptions of such vocabularies are called speech-recognition grammars. <SimpleMenu>s and <SimpleFormSelect>s provide such grammars with their lists of <SimpleMenuItem>s and <SimpleFormOption>s. However, in examples such as the one above, the system should be listening for an arbitrary number. This is indicated by the type attribute of <SimpleFormItem>, as follows.

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
```

```
<SimpleResult>
 <SimpleContainer>
  <SimpleForm target="guess.jsp">
   <SimpleFormItem name="guess" type="number">
    <SimpleTitle>
    I am thinking of a number between 1 and 100.
    What is your first guess?
    </SimpleTitle>
   </SimpleFormItem>
  </SimpleForm>
 </SimpleContainer>
</SimpleResult>
```

Setting type="number" tells the system to listen for any utterance that corresponds to a spoken number, if such an utterance is heard, then assigns the corresponding number to the identifier "guess". In addition to number, the values boolean, digits, date, time, currency, and phone also specify vocabularies to listen for. Besides specifying the type attribute, the developer can enhance the voice features by observing the following guidelines:

- The voice experience can be enhanced with prerecorded audio using the <SimpleAudio> element.
- As confirmation, echo the recognized utterance and allow the user to cancel if an input has been misrecognized.
- Always provide context-sensitive help.
- As necessary, use the deviceclass attribute to tailor audio and text messages to voice (but use this attribute sparingly, as it tends to obfuscate the markup).
- Always provide the user the option of continuing in a service by "moving forward" -- providing an appropriate command leading to the place the user wants to go -- rather than forcing them to "back out" using cancel.
- Provide special event handlers for recognition failures (noinput, nomatch) and Internet fetch failures (error.badfetch) where appropriate.

The following example improves the user experience through the implementation of these guidelines.

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
  <SimpleForm target="tipcalc.jsp">
```

```
   <SimpleFormItem name="howmuch" type="currency">How much is the bill?
   </SimpleFormItem>
   <SimpleFormItem name="howmany" format="N*" type="number">
   How many are in your party?
    <SimpleCatch type="cancel">Canceling.
     <SimpleClear name="howmuch"/>
    </SimpleCatch>
   </SimpleFormItem>
   <SimpleFormSelect name="howbig" deviceclass="microbrowser pdabrowser
pcbrowser micromessenger messenger">
    <SimpleTitle>How big do you want your tip to be?</SimpleTitle>
    <SimpleFormOption value="10">small (10%)</SimpleFormOption>
    <SimpleFormOption value="15">medium (15%)</SimpleFormOption>
    <SimpleFormOption value="20">large (20%)</SimpleFormOption>
   </SimpleFormSelect>
   <SimpleFormSelect name="howbig" deviceclass="voice" autoprompt="false">
    <SimpleTitle>
     How big do you want your tip to be?
     For 'ten percent' say 'small',
     for 'fifteen percent' say 'medium',
     for 'twenty percent' say 'large'.
    </SimpleTitle>
    <SimpleFormOption value="10">small</SimpleFormOption>
    <SimpleFormOption value="15">medium</SimpleFormOption>
    <SimpleFormOption value="20">large</SimpleFormOption>
    <SimpleCatch type="nomatch">Please say that again</SimpleCatch>
    <SimpleCatch type="cancel">Canceling.
     <SimpleClear name="howmuch"/>
     <SimpleClear name="howmany"/>
     <SimpleClear name="howbig"/>
    </SimpleCatch>
   </SimpleFormSelect>
  </SimpleForm>
 </SimpleContainer>
</SimpleResult>
```

6

Advanced User Interactions and Channel Optimization

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

- Section 6.1, "Introduction"
- Section 6.2, "Events and Tasks Using SimpleBind"
- Section 6.3, "Device Headers and Device Class"

6.1 Introduction

In this chapter, we will discuss some of the advanced user interaction techniques provided by Oracle9*i*AS Wireless. So far, we have seen how Oracle9*i*AS Wireless allows users to specify a task when a user performs an action (for example, pressing a soft key on the phone or uttering a command on a voice enabled device). Advanced User Interactions provide the ability to perform many tasks in response to an action triggered by a user whenever supported by the device. And, the ability to perform tasks based on the value input by users is highly desirable.

Oracle9*i*AS Wireless provides an elaborate scheme to facilitate very sophisticated binding of tasks and actions. This is performed by the SimpleBind element which may appear in the context of SimpleText, SimpleForm, SimpleFormItem, SimpleFormSelect, SimpleMenu, SimpleResult or SimpleContainer.

6.2 Events and Tasks Using SimpleBind

SimpleBind lets you specify SimpleTask which is performed in response to an action specified as the child of SimpleMatch element. SimpleMatch may specify primary1, primary2 etc. keys, nospeech, noinput, or an item of SimpleMenu conditions, etc. Only one task may be specified in SimpleMatch and when this action is performed,

all the tasks specified in SimpleTask are performed. SimpleTask may also perform tasks selectively by using SimpleSwitch, SimpleCase and SimpleDefault elements which are analogous to the switch and case constructs of many programming languages.

In SimpleSwitch, a value of a particular user input is compared to the values enumerated by SimpleCase elements. SimpleTask may specify to:

- go to a remote location using SimpleGo
- display a text item using SimpleTextItem
- refresh the device screen (if supported) using SimpleRefresh
- clear the specified device form fields using SimpleClear and SimpleName
- allow voice users to reprompt input using SimpleReprompt
- exit the application using SimpleExit
- disconnect the device from connected state (such as a voice browser) using SimpleDisconnect
- define back operation using SimplePrev and SimpleGo
- submit a form using SimpleSubmit.

The rendering characteristics of the SimpleBind element are specified by the SimplDisplay element. SimpleDisplay supports SimpleTextItem as child elements that contain the actual render and display content. This allows you to play an audio or render the text for a MenuItem. See example SimpleBindExample.xml.

6.2.1 SimpleBind.xml

```
<SimpleBind deviceclass="voice microbrowser">
          <SimpleMatch>
            <SimpleFinish/>
            <SimpleGrammar>
              yes {}| correct {}| true {} | one {}
            </SimpleGrammar>
            <SimpleDTMF>1</SimpleDTMF>
          <SimpleKey type="primary"/>
          </SimpleMatch>

          <SimpleTask>
            <SimpleSubmit
              target="changepin.jsp"
              name="Submit"
```

```
      method="post">
      <SimpleName name="p_old_pin" />
      <SimpleName name="p_new_pin" />
    </SimpleSubmit>
  </SimpleTask>

  <SimpleDisplay>
    <SimpleTextItem deviceclass="voice">
      <SimpleAudio src="sayYesOrPressOne.wav">
        say yes, or press one, to submit
      </SimpleAudio>
    </SimpleTextItem>

    <SimpleTextItem deviceclass="microbrowser">
      Submit
    </SimpleTextItem>
  </SimpleDisplay>
</SimpleBind>
```

Figure 6–1 Results of SimpleBind, SimpleMatch and SimpleDisplay

6.2.2 Device Specific SimpleBind

SimpleBind is primarily useful while writing voice applications. However, an application may use SimpleBind based on a particular device by the use of the

'deviceclass' attribute. This attribute can take the values 'pdabrowser', 'pcabrowser', 'voice', 'microbrowser', 'micromessenger' and 'messenger'.

6.3 Device Headers and Device Class

Devices are classified based on two criteria in Oracle9*i*AS Wireless. The classification is based on:

- form factor of the device
- communication channel of the device (synchronous request / response or async mode)

See the following document for more information:

http: / / mobile.us.oracle.com / ompm / site / internal / api / iaswheaders / dheaders.jsp

Developers may develop value added services, which make use of device specific properties. For example, Oracle9*i*AS Wireless does not support server side management of large response. A service may use the maximum size of response for a device to provide navigation dynamically. The following headers are supported:

- **X-Oracle-Device.Class:** Indicates the channel mode and the form factor of a device. Each value of the Device.Class indicates a unique communication channel mode and the unique form factor. The value set for the attribute "deviceclass" is same as the header *X-Oracle-Device.class*. Note that device.class does not represent target device markup language.
- **X-Oracle-Device.Orientation:** Indicates the orientation of a device. May be used by an application to change the rendering style for certain devices. Possible values are "*landscape*" "*portrait*". Default value is "*portrait*".
- **X-Oracle-Device.MaxDocSize:** Approximate value of maximum number of bytes of content that can be handled by the device in question. The approximation arises due to fact that Oracle9*i*AS Wireless XML size may not be the same as transformed device-specific markup language. If the service returns a Oracle9*i*AS Wireless XML document greater than the MaxDocSize, the response for such a request is unspecified. It is not guaranteed that a document size bounded by MaxDocSize will result in the content size, which can be pushed to the device. The value of the parameter is set by the administration tool of Oracle9*i*AS Wireless for the deviceclass. The default value is 0.

- **X-Oracle.Device.Secure:** Indicates if the connection between the Oracle9*i*AS Wireless server and the device was secure when the current request for the resource was made. Possible values are "*true*" or "*false*".

The following jsp uses a PageNavigation bean to deliver news content in multiple trips.

6.3.1 Article.jsp

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<%@ page import="oracle.wireless.xmldevguide.PageNavigation"%>
<%
boolean loopback = Boolean.valueOf(request.getParameter("loopback")).booleanValue();
int pageIndex = 0;
try {
    pageIndex = Integer.parseInt(request.getParameter("pageIndex"));
}
catch(Exception ex){}
%>
<SimpleResult>
<SimpleContainer>
    <jsp:useBean id="contentHandler" class="oracle.wireless.xmldevguide.PageNavigation"
scope="session"/>
    <%
    if(!loopback) {
        String size = request.getHeader("X-Oracle-Device.MaxDocSize");
        if(size != null && !("0".equals(size))) {
           contentHandler.setDeckSize(Integer.parseInt(size));
        }
        pageIndex = 0;
        // get the article content from a source.
        String articleContent = "OracleMobile Online Studio is an online "+
        "developer portal for quickly building, testing and deploying "+
        "wireless applications. It lets any developer, systems integrator "+
        "or independent software vendor quickly develop a mobile application "+
        "that is immediately accessible from all devices. This unique, next "+
        "generation environment allows companies to benefit from faster time "+
        "to market, increased productivity, and a dramatically simplified "+
        "testing cycle, while providing access to the latest mobile applications "+
        "and tools. It enables you to focus on your business logic which is your "+
        "core competency, while we focus on the device complexity, our core "+
        "competency. <SimpleBreak/><SimpleBreak/>"+
        "OracleMobile Online Studio's build, test, and deploy model is new and "+
        "unique to software development. It presents a hosted approach to developing "+
        "dynamic content. You do not need to download any software or tools to start "+
        "using it. All you need to do is access the OracleMobile Online Studio, "+
```

```
        "register, and login. Once authenticated, you will have access to "+
        "reusable modules, examples, documentation, runtime information, and other "+
        "useful resources.  <SimpleBreak/><SimpleBreak/>"+
        "Now you can even use OracleMobile Online Studio to write a single application "+
        "that can be accessed via both wireless and voice interfaces. Listen to your "+
        "OracleMobile Online Studio applications by calling: "+
        "888-226-4854. <SimpleBreak/><SimpleBreak/>"+
        "Simplify the development of your OracleMobile Online Studio application "+
        "with Where2Net's daVinci Studio.";

         contentHandler.setContent(articleContent);
    }
    String nextURL = null;
    String previousURL = null;
    int numPages = contentHandler.getAvailablePages();
    if(numPages > 1) {
        nextURL = (pageIndex < numPages - 1) ?
"article.jsp?loopback=true&amp;pageIndex="+(pageIndex + 1) : null;
        previousURL = (pageIndex > 0) ? "article.jsp?loopback=true&amp;pageIndex="+
(pageIndex - 1) : null;
    }
    String articleTitle = (pageIndex == 0) ? "OracleMobile online studio" : "contd...";
    %>
    <SimpleText>
     <SimpleTitle><%=articleTitle%></SimpleTitle>
    <%
        String s = (nextURL == null) ? "articleIndex.jsp" : nextURL;
        if(pageIndex != numPages - 1) {
            %>
            <SimpleAction type="primary2" label="Close" target="articleIndex.jsp"/>
            <SimpleAction type="primary1" label="Next" target="<%=s%>"/>
            <%
        }
        else {
            %>
            <SimpleAction type="primary1" label="Close" target="<%=s%>"/>
            <%
        }
        %>
        <SimpleTextItem><%=contentHandler.getPage(pageIndex)%></SimpleTextItem>
        <%
        if(previousURL != null) {
            %>
            <SimpleTextItem><SimpleHref
target="<%=previousURL%>">Previous</SimpleHref></SimpleTextItem>
            <%
            }
            if(nextURL != null){
                %>
                <SimpleTextItem><SimpleHref
```

```
target="<%=nextURL%>">Next</SimpleHref></SimpleTextItem>
                <%
            }
            %>
    </SimpleText>
    </SimpleContainer>
 </SimpleResult>
```

6.3.2 PageNavigation.java

```
package oracle.wireless.xmldevguide;

import java.io.StringReader;
import java.io.StringWriter;
import java.io.Serializable;
import java.io.IOException;

import java.util.ArrayList;

/**
 * The bean breaks a text content into mutiple deck of a defined size. Content
 * deck do not include any formatting information of the content which should
 * be provided by the content view.
 *
 * @author Chandra Patni
 * @version 1.0
 */
public class PageNavigation implements Serializable {

    /**
     * To keep the location of a page
     */
     private class Page {
        /**
         * starting index of the page, inclusive of start
         */
        public int start;
        /**
         * end index of the page, exclusive
         */
        public int end;
        /**
         * returns the length of the page
         */
        public int length() {
```

```
            return end - start;
        }

        /**
         * retruns the content of the page
         */
        public String toString() {
            return content.substring(start, end);
        }
     }

    /**
     * Default size of a deck in characters. The actual deck size will be
adjusted
     * so that a word is not split. However, an orphan, end of paragraph etc
     * conditions are not checked for.
     */
    public static final int DECK_SIZE = 900;

    /**
     * size of a deck. default value is 900 chars
     */
    private int deckSize = DECK_SIZE;

    /**
     * Sets the size of one deck. Should be called before setContent()
     */
    public void setDeckSize(int value) {
        deckSize = value;
    }

    /**
     * Returns the size of one deck.
     */
    public int getDeckSize() {
        return deckSize;
    }

    /**
     * Conent to be decked
     */
    private String content;

    /**
     * Pages in the content
```

```
 */
private Page pages[];

/**
 * The total number of pages by the content
 */
private int totalPages;

/**
 * Default constructor
 */
public PageNavigation() {
}

/**
 * Default constructor
 */
public PageNavigation(String content) {
    setContent(content);
}

/**
 * get the page content at the given index
 */
public String getPage(int index) {
    return pages[index].toString();
}

/**
 *  Returns the total number of pages
 */
public int getAvailablePages() {
    if(pages == null) return 0;
    return pages.length;
}

/**
 * initializes the bean
 */
private void init() {
    // get the rough estimate of pages
    totalPages = content.length() / deckSize + 1;
    // initialize the array
    int lastIndex = 0;
    ArrayList list = new ArrayList(totalPages);
```

```
        Page p = null;
        while((p = getNextPage(lastIndex)) != null) {
            list.add(p);
            lastIndex = p.end;
        }
        pages = (Page []) list.toArray(new Page[list.size()]);
    }

    private Page getNextPage(int lastIndex) {
        if(lastIndex >= content.length()) return null;
        char c = content.charAt(lastIndex);
        while(Character. isWhitespace(c)) {
            if(++lastIndex >= content.length()) return null;
            c = content.charAt(lastIndex);
        }
        Page p = new Page();
        p.start = lastIndex;
        // again look for whitespaces while trimming the content.
        p.end = p.start + deckSize;
        if(p.end >= content.length()) {
            p.end = content.length();
            return p;
        }
        // if not then we need to figure out the previous white space
        do {
            c = content.charAt(p.end);
            if(Character. isWhitespace(c)) {
                return p;
            }
            p.end--;
            if(p.end == 0) {
                p.end = p.start + deckSize;
                return p;
            }
        }while(true);
    }

    /**
     * sets the content to the specified value. default MIME type is text/plain
     */
    public void setContent(String s) {
        content = s;
        init();
    }
}
```

7

Mobile Modules

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

- Section 7.1, "Introduction"
- Section 7.2, "Wireless XML Attributes for Mobile Modules"
- Section 7.3, "Shipped Mobile Modules"
- Section 7.4, "Using Shipped Mobile Modules"
- Section 7.5, "Developing Custom Mobile Modules"

7.1 Introduction

Mobile Modules are wireless services with well-known virtual URL (OMP URL, i.e. omp:/ / my.module). Mobile Modules provide an analogous mechanism to data abstraction and interfaces. They allow a component-based programming model for building mobile applications within the Oracle9*i*AS Wireless framework. Component-based programming provides rapid application development, reusable components and easy-to-maintain code which are essential to timely, successful deployment of web applications.

Mobile Modules can be called from any application or module and may be instructed to return control to another application or module. Calls may be nested to any level. This mechanism of bi-directional linking allows quick applications assembly.

Important difference between a module and a regular service is that the module receives information about the service it needs to return to after it is done. This is not always the caller of the module (the module caller may want the module to return to a different service).

An example of an application that leverages Mobile Modules could be a store locator application for a retail company. A developer writing this application could improve the interface by linking to the Location Mobile Module, which enables a user to store frequently accessed locations as landmarks. The application would then offer to find the nearest store based on one of those locations, saving the user the time and effort of entering an address. The next logical step would be to link to the Driving Directions Mobile Module, so that a customer could easily get directions to the store they have selected. This would enable the user to get directions without typing in any additional information, since both the starting location and the destination address (store) would intelligently populate the corresponding fields in the application.

7.2 Wireless XML Attributes for Mobile Modules

The *target* attribute of *SimpleMenuItem*, *SimpleAction*, *SimpleHref*, and *SimpleForm* may be used for linking to a Mobile Module. The value of the target attribute starts with *omp://* for accessing modules.

> **Note:** The value of the omp:// URL is not important. There are only two important things that you need to keep in mind:
>
> The value must start with omp://
>
> The value must be unique (just like an http:// URL)

These are the XML attributes that are used for linking to Mobile Modules:

- **target** - is the only mandatory attribute. Its value is the virtual (omp://) URL for the Mobile Module.
  For example:

  ```
  <SimpleMenuItem target="omp://oracle.com/module"
  callbackurl="%value service.home.url%">Call My Mobile
  Module</SimpleMenuItem>
  ```

- **secure** - an optional attribute. The value is either "true" or "false". This attribute is used to switch between HTTP (secure="false") and HTTPS (secure="true") protocol for the connection between the end user device and Oracle9*i*AS Wireless server when calling the module. If you do not set this attribute then the current protocol will be used.
  For example:

```
<SimpleMenuItem target="omp://oracle.com/module"
callbackurl="%value service.home.url%" secure="false">Call
My Mobile Module</SimpleMenuItem>
```

- **callbackurl** - This is the URL of the service where the mobile module should return after it is done. The default value is the current caller service. You can use Mobile Context (see Chapter 8, "XML Tag Glossary") to specify values for the attribute.
  For example:

  ```
  <SimpleMenuItem target="omp://oracle.com/module"
  callbackurl="%value service.home.url%"> Call My Mobile
  Module</SimpleMenuItem>
  ```

- **callbacksecure** - an optional attribute. The value is either "true" or "false". This attribute is used to switch between HTTP (callbacksecure="false") and HTTPS (callbacksecure="true") protocol for the connection between the end user device and Oracle9*i*AS Wireless server when the module calls back.
  For example:

  ```
  <SimpleMenuItem target="omp://oracle.com/module"
  secure="true" callbackurl="%value service.home.url%"
  callbacksecure=”false”> Call My Mobile
  Module</SimpleMenuItem>
  ```

- **callbackparam** - an optional attribute that sets parameters to be passed to the caller after the module is done.
  For example:

  ```
  <SimpleMenuItem target="omp://hostname/module"
  callbackurl="%value service.home.url%"
  callbackparam="foo=bar&amp;test=TEST&amp;a=z ">My Mobile
  Module</SimpleMenuItem>
  ```

7.3 Shipped Mobile Modules

Oracle9*i*AS Wireless contains a set of 17 ready-to-use modules subdivided in the following areas: mobile commerce, PIM and location-based services. Application developers may reuse these modules as the jumpstart of their wireless development work, or develop their own modules, by following the instructions in this document. For a complete reference on the shipped Mobile Modules, see Chapter 18, "Mobile PIM and eMail" and Chapter 19, "m-Commerce"

7.4 Using Shipped Mobile Modules

7.4.1 Commerce Services

To use the ***Payment Module*** in order to make an credit card payment of US$ 90.00 you will use:

```
<SimpleMenuItem
target="omp://oracle/services/commerce/payment?AMOUNT=90&merch
antid=bookshop&MODE=ONLINE&TYPE=AUTH&INSTRTYPE=CC"
callbackurl="%value service.home.url%">Pay
amount</SimpleMenuItem>
```

The payment module will take the action after the user chooses this menu, and will present a flow of cards that will lead to the payment itself. In the end the Payment Module will return the transaction id in the HTTP request.

7.4.2 PIM Services

To use the ***Mail Module*** you will need to inform the action and the email to whom you want to sent the message:

```
<SimpleMenuItem
target="omp://oracle/services/pim/mail?action=messageto&mailto
=jsmith@company.com" callbackurl="%value
service.home.url%">Send eMail</SimpleMenuItem>
```

7.4.3 Location Services

To use the ***Maps Module*** you will need to inform the address you want to map:

```
<SimpleMenuItem
target="omp://oracle/services/location/maps?FL=500 Oracle
Parkway&CI=Redwood Shores&ST=CA" callbackurl="%value
service.home.url%">Map Oracle</SimpleMenuItem>
```

Note: These are just small examples on how to call the shipped Mobile Modules. For a more complete reference of the Modules OMP URLs, input and output values please see Chapter 18, "Mobile PIM and eMail" and Chapter 19, "m-Commerce".

7.5 Developing Custom Mobile Modules

Developing Mobile Modules is not very different than developing your own services. For more details about how to develop service see Chapter 15, "Using Location Services", Chapter 18, "Mobile PIM and eMail", and Chapter 19, "m-Commerce". In our examples we are going to use the HttpAdapter. The mobile modules will use simple JSP pages.

7.5.1 "Hello World" Mobile Module

Our first mobile module does not do much. It will just display "Hello World" on the end user device and a link to go back to the module caller service.

7.5.1.1 Create and publish the JSP pages for the module and the caller services

Here is the JSP code for the module:

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1.0//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<%@ page contentType="text/vnd.oracle.mobilexml; charset=UTF-8" %>
<%@ page language="java" %>
<%@ page session="false" %>
<SimpleResult>
  <SimpleContainer>
    <SimpleMenu>
      <SimpleTitle>Hello World</SimpleTitle>
      <SimpleMenuItem target="%value module.callback.url%">Go Back To The
Caller</SimpleMenuItem>
    </SimpleMenu>
  </SimpleContainer>
</SimpleResult>
```

Please save this code in HelloWorldModule.jsp and publish it at let say http://localhost/jsp/HelloWorldModule.jsp.

And the JSP code for the caller service:

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1.0//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<%@ page contentType="text/vnd.oracle.mobilexml; charset=UTF-8" %>
<%@ page language="java" %>
<%@ page session="false" %>
<SimpleResult>
```

```
  <SimpleContainer>
    <SimpleMenu>
      <SimpleTitle>Hello World Caller</SimpleTitle>
      <SimpleMenuItem target="omp://HelloWorld" callbackurl="%value
service.home.url%">Call Hello World Module</SimpleMenuItem>
    </SimpleMenu>
  </SimpleContainer>
</SimpleResult>
```

Please save this code in HelloWorldCaller.jsp and publish it at (for instance): http://localhost/jsp/HelloWorldCaller.jsp

7.5.1.2 Create HelloWorldModuleMS and HelloWorldCallerMS MasterServices

After we publish the JSP pages we need to create two HttpAdapter based MasterServices. Use the Service Designer web tool to do that. See *Oracle9iAS Wireless Getting Started and System Guide* for more details about creating MasterServices

.

> **IMPORTANT:** Please mark the HelloWorldModuleMS MasterService as *"Modulable".*

7.5.1.3 Create the caller and the module services

After you are done with the MasterServices you need create two services: HelloWorldModule and HelloWorldCaller. Use the Content Manager web tool to do that. See *Oracle9iAS Getting Started and System Guide* for more details about creating MasterServices.

> **IMPORTANT:** The type of the HelloWorldCaller service should be *"Normal Service"*. The type of the HelloWorldModule service should be "*Module*". Set the OMP URL for the HelloWorldModule service to *omp://HelloWorld.*
>
> Before you can test the newly created services you need to assign them to a Group so the users in that group can invoke those services.

That is it. Now you can test the two services from your device.

7.5.2 Sending Parameters to a Mobile Module

The Mobile Modules that you want to develop will most likely take some input from its caller and then return something back after there are done. Below are the JSP pages that show how a caller service can send an input parameter to a module. Publishing those two JSP pages on Oracle9*i*AS Wireless is the same as publishing the previous JSP pages.

Here is the code for the HelloNameModule.jsp

```
<?xml version = "1.0" encoding = "ISO-8859-1" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1.0//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<%@ page contentType="text/vnd.oracle.mobilexml; charset=ISO-8859-1" %>
<%@ page language="java" %>
<%@ page session="false" %>
<%
   String uname = request.getParameter("uname");
%>
<SimpleResult>
   <SimpleContainer>
    <SimpleMenu>
      <SimpleTitle>Hello Module Says Hello <%=uname%></SimpleTitle>
      <SimpleMenuItem target="%value module.callback.url%">Go Back To The
Caller</SimpleMenuItem>
    </SimpleMenu>
   </SimpleContainer>
</SimpleResult>
And the JSP code for the HelloNameCaller.jsp:
<?xml version = "1.0" encoding = "ISO-8859-1" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1.0//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<%@ page contentType="text/vnd.oracle.mobilexml; charset=ISO-8859-1" %>
<%@ page language="java" %>
<%@ page session="false" %>
<SimpleResult>
   <SimpleContainer>
      <SimpleForm target="omp://HelloName">
         <SimpleTitle>Please Enter User Name</SimpleTitle>
   <SimpleFormItem name="uname" />
      </SimpleForm>
   </SimpleContainer>
</SimpleResult>
```

IMPORTANT: When you create the Oracle9*i*AS Wireless MasterService and service objects remember to:

mark the HelloNameModuleMS MasterService as "*Modulable*"

set the type of HelloNameCaller service to be "*Normal Service*" and the type of the HelloNameModule service to "*Module*"

Set the OMP URL for the HelloWorldModule service to ***omp://HelloName***

Note: Before you can test the newly created services you need to assign them to a Group so the users in that group can invoke those services.

That is it. Now you can test the two services from your device.

8

XML Tag Glossary

This document present the following topic:

The XML DTD defines the abstract device markup language used in the OracleMobile Online Studio application framework. The goal of the definitions is to be a superset of the markup languages for a variety of devices. Elements in the DTD represent elements of an abstract user interface which translate to device-specific formats. The following is a list of tag elements in the XML DTD. See *Oracle Technology Network* for more details, and to view the tag element tree. The XML is derived from the DTD of Oracle9*i*AS Wireless with hosting extensions added. Note that the tag names are case-sensitive as below.

8.1 XML Tags

SimpleAction

This tag provides the ability to define a link or submit action that navigates the user to a new context. Mobile devices can associate a submit action to a number of input methods (of the device), like pressing a Key on wap devices or saying a command on voice enabled devices. Action can have SimpleTextItem as child, this is used for rendering of Action tag in voice.

Table 8–1 SimpleAction Tag

Name	Description	Value(s)	Default Value
name	Name identifier for the element instance.	CDATA	OPTIONAL
callbackurl	ASW Module Support. Indicates the URL to return back if the current action leads the user into a different application (application implementing Wireless Module functionary).	CDATA	OPTIONAL
callbackparam	ASW Module Support. Indicates the return parameters of the callbackurl. When Module returns the context back to the callee application, the callbackparam is passed back for the callee to construct its application state.	CDATA	OPTIONAL
callbacksecure	Indicates the mode of communication, when callback occurs, between Wireless server and the device. Setting callbacksecure="true" will enable a secure connect mode between Wireless and the device when the module performs a callback (to the callbackurl). If not specified, the connect mode will be based on the current request mode.	(true | false)	OPTIONAL
target	URI to navigate to when action is activated. This URL is always rewritten by the Server to point back to Wireless Server, except when mimetype attribute not "text/vnd.oracle.mobilexml". Also supports "callto:" for Phone call and "mailto:" for email support.	CDATA	OPTIONAL
mimetype	mime-type of target URI. Lets the Wireless server know the target resources mime-type. If the target mime-type is not text/vnd.oracle.mobilexml, the Wireless server will not rewrite the URL.	CDATA	ext/vnd.oracle.mobilexml OPTIONAL

Table 8–1 SimpleAction Tag

Name	Description	Value(s)	Default Value
static_target	URI to navigate to when action is activated. This URL is never rewritten by server. If exists, this will override the "target" attribute. Also supports "callto:" for Phone call and "mailto:" for email support.	CDATA	OPTIONAL
secure	Indicates the mode of communication between Wireless server and the device. Setting secure="true" will enable a secure connect mode between Wireless and the device for the specified target. If not specified, the connect mode will be based on the current request mode. This DOES NOT indicate mode of connection between Wireless and the remote content source (the service), Wireless will connect to the remote service based on the protocol specified in the target attribute ("http" vs. "https")	(true | false)	OPTIONAL
fetchaudio	Voice only attribute. The URI of an audio clip to play while the "target" is being fetched.	CDATA	OPTIONAL
type	Defines the type of Binding in the target device. Can take any string value. Continue, primary, secondary are special types. "primary" and "secondary" map to the primary and secondary keys resp. Continue is a special Primary key, tells the Voice service to continue without waiting for the user. If both continue and primary are defined both of them will map to the primary key. The following type will also be support by the transformers for backward compatibility "accept"|"soft1"|"option1"|"option2".	(continue | primary | secondary)	REQUIRED
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL
icon	Built-in icon name for HDML/WML. Will be used if specified.	CDATA	OPTIONAL

Table 8–1 SimpleAction Tag

Name	Description	Value(s)	Default Value
src	The URL to an image. Image from will be displayed. (In SimpleAction / Href this images needs to be used instead of the label.	CDATA	OPTIONAL
addImageExtension	Allows the server to use the right image format from a list of available formats. Based on the available images from the app (specified by the "available" attribute) and based on the device browser capability the server will pick the right image to be used. For example: If image is "oracle" and available is set to "jpg gif wbmp", server will use "oracle.wbmp" in WML (Phone.com) browser, "oracle.gif" for a HTML browser.	(true | false)	true OPTIONAL
available	Application can specify a list of available image formats, for example: available = "jpg gif g2.gif wbmp bmp" (g2.gif indicates a grayscale depth 2 image, for devices like Palm). This allows the server to use the correct image format supported by device (based on device browser properties).	CDATA	OPTIONAL
label	Label for action button, displayed when action is bound to a button on a visual device	CDATA	OPTIONAL
dtmf	digit to be pressed on phone or DTMF tone. dtmf attribute just takes one value (a simplified form of voice SimpleDTMF tag). Will work on wap devices, if supported by the device.	CDATA	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Usage

SimpleAction, for a given type, conforms to scopings rule (like the programming languages do).

If SimpleAction is defined as a child of SimpleMenu and also as an child of the enclosing SimpleContainer for a given "type" of SimpleAction, the SimpleAction tag within the Menu overrides the SimpleAction of the SimpleContainer. (Note: If the value for "type" attribute is different then the two SimpleAction's will active within the context. Also if two SimpleActions are defined with same "type" value within same context, then the action is undefined).

The "type" attribute can be used to define device specific actions, along with the deviceclass attribute.

Related Tags

Table 8–2 SimpleAction Related Tags

Parents	Children
SimpleText	SimpleCache
SimpleForm	SimpleTextItem
SimpleFormItem	SimpleGrammar
SimpleFormSelect	SimpleDTMF
SimpleContainer	
SimpleResult	
SimpleMenu	

SimpleAudio

Plays an audio file on voice devices.

Table 8–3 SimpleAudio Tag

Name	Description	Value(s)	Default Value
src	The URL to an audio source file. Used in to play audio (Voice).	CDATA	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). This tag is applicable for Voice devices only. Will not be supported on other devices even if specified.	CDATA	OPTIONAL

Related Tags

Table 8–4 SimpleAudio Related Tags

Parents	Children
SimpleTextItem	[PCDATA]
SimpleSpan	SimpleCache
SimpleUnderline	SimpleBreak
SimpleImage	SimpleEm
SimpleHref	SimpleStrong
SimpleStrong	SimpleSpan
SimpleEm	SimpleUnderline
SimpleAudio	SimpleAudio
SimpleSpeech	SimpleSpeech
SimpleTitle	SimpleImage
	SimpleValue

SimpleBind

Bind is an extended action tag that can be invoked by multiple events and perform multiple tasks. A task / action can be triggered by events like by device keys (touch tone), by voice commands or by selecting a MenuItem. The action in turn may comprise of set of tasks to perform on match of the event. For e.g. Submit a form can happen when user clicks submit button, or when the user presses a key on the device or just says "submit" on the voice devices. SimpleBind tag defines all these events that needs to matched. SimpleBind tag also defines to set of actions that are mapped to event matches (SimpleMatch). The action can include displaying a flash screen to inform the user of the submit action and then actually submitting the form (2 actions for one set of events). SimpleBind also encloses the SimpleDisplay tag. SimpleDisplay tag is the rendering component of the SimpleBind, Indicates how the Bind action is rendered on the device. This rendering is used to render binding of type menu item and actions. For example: SimpleDisplay can include a SimpleTextItem saying "press or say 1 to submit".

Table 8–5 SimpleBind Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–6 SimpleBind Related Tags

Parents	Children
SimpleText	SimpleMatch
SimpleForm	SimpleTask
SimpleFormItem	SimpleDisplay
SimpleFormSelect	
SimpleContainer	

Table 8–6 SimpleBind Related Tags

Parents	Children
SimpleResult	
SimpleMenu	

SimpleBreak

Creates a break, new line on text devices and pause on voice devices.

Table 8–7 SimpleBreak Tag

Name	Description	Value(s)	Default Value
msecs	milliseconds in duration of the break for voice devices.	xsd:nonNegativeInteger	OPTIONAL
rule	Generate an HR with this break (HTML)	(true | false)	false
			OPTIONAL
size	size of the break (VoiceXML)	(none | small | medium | large)	medium
			OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–8 SimpleBreak Related Tags

Parents	Children
SimpleTextItem	none
SimpleSpan	
SimpleUnderline	
SimpleImage	
SimpleHref	
SimpleStrong	
SimpleEm	
SimpleAudio	

Table 8–8 SimpleBreak Related Tags

Parents	Children
SimpleSpeech	
SimpleTitle	
SimpleHelp	

SimpleCache

An URL can be cache on the gateway (like WAP gateway), client or both of them. Also used when a URL needs to be prefetched while still showing the current content (only supported devices in certain devices). SimpleCache defines all these policies.

Table 8–9 SimpleCache Tag

Name	Description	Value(s)	Default Value
timeout	Time, in milliseconds, to wait while fetching a resource before failing.	xsd:nonNegativeInteger	OPTIONAL
policy	Cache on gateway, client, both or none. Value of "public" indicates cache can on the Gateway (like the WAP), "private" indicates client only cache.	public | private | both | none)	private OPTIONAL
prefetch	prefetch policy. Certain devices can prefetch a "target" resources, before the user requests for the resources. The attribute controls the policy of such a prefetch-able resource.	(onload | safe | streamed)	safe OPTIONAL
ttl	Time to live for cached data in milliseconds	xsd:nonNegativeInteger	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–10 SimpleCache Related Tags

Parents	Children
SimpleImage	none
SimpleHref	
SimpleAction	

Table 8–10 SimpleCache Related Tags

Parents	Children
SimpleAudio	
SimpleGrammar	
SimpleDTMF	
SimpleGo	
SimpleMenuItem	
SimpleResult	

SimpleCase

Tag to write case statements within a SimpleSwitch tag. This allows the developer to perform client side "actions" on devices. Support for Switch / Case is not universal. Is Supported only in Wap (HDML / WML) and Voice (VoiceXML) devices.

Table 8–11 SimpleCase Tag

Name	Description	Value(s)	Default Value
value	The value of for the Case statement (to be compared with the value of form field, identified by the name attribute of SwitchCase)	CDATA	REQUIRED
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–12 SimpleCase Related Tags

Parents	Children
SimpleSwitch	SimpleGo
	SimpleTextItem
	SimpleRefresh
	SimpleClear
	SimpleReprompt
	SimpleExit
	SimpleDisconnect
	SimplePrev
	SimpleSubmit

SimpleCatch

Catches an event. Voice only tag. This can be used to capture predefined voice events like "noinput" "exit" etc. and perform actions on it. For example: on "noinput" (formitem) event the user can be given some help instructions and be reprompted for the input. Events include errors generated (Errors are also an instance of event)

Table 8–13 SimpleCatch Tag

Name	Description	Value(s)	Default Value
count	The occurrence of the event (default is 1). The count allows you to handle different occurrences of the same event differently For example:. Need to give extra help messages if the user says "help" twice for the same form item. The form / formitem / menu etc. (where ever SimpleCatch can occur) maintain a counter for each event that occurs while it is being visited, these counters are reset each time the form is re-entered.	Positive Int	1 OPTIONAL
type	Predefined Voice events. Possible values include cancel, error, exit, help, noinput, nomatch, telephone.disconnect.	CDATA	REQUIRED
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–14 SimpleCatch Related Tags

Parents	Children
SimpleText	[PCDATA]
SimpleForm	SimpleGo
SimpleFormItem	SimpleTextItem
SimpleFormSelect	SimpleRefresh

Table 8–14 SimpleCatch Related Tags

Parents	Children
SimpleResult	SimpleClear
SimpleMenu	SimpleReprompt
	SimpleExit
	SimpleDisconnect
	SimplePrev
	SimpleSubmit

SimpleClear

Clears a list of client side form fields identified by the Name list (SimpleName). Works on WML / Voice (Voice) device only. Useful in voice applications e.g. Clearing a form field in voice will allow the Voice engine to reprompt the User for the form field again.

Table 8–15 SimpleClear Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–16 SimpleClear Related Tags

Parents	Children
SimpleCatch	SimpleName
SimpleCase	
SimpleTask	

SimpleCol

Defines a column of a table.

Table 8–17 SimpleCol Tag

Name	Description	Value(s)	Default Value
bgcolor	background color	CDATA	OPTIONAL
rowspan	from HTML table spec	CDATA	OPTIONAL
colspan	from HTML table spec	CDATA	OPTIONAL
bordercolor	from HTML table spec	CDATA	OPTIONAL
height	cell height	CDATA	OPTIONAL
width	cell width	CDATA	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL
valign	vertical alignment	(top | middle | bottom)	top OPTIONAL
halign	horizontal alignment	(left | center | right)	left OPTIONAL
wrapmode	text wrap mode	(wrap | nowrap)	wrap OPTIONAL

Related Tags

Table 8–18 SimpleCol Related Tags

Parents	Children
SimpleTableHeader	SimpleTextItem

Table 8–18 SimpleCol Related Tags

Parents	Children
SimpleRow	

SimpleContainer

The root element that contains all major block constructs like form, menu and Text.

Table 8–19 SimpleContainer Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL
id	ID attribute of the element. Used for Navigation within a XML response (target="#ID")	xsd:ID	OPTIONAL

Related Tags

Table 8–20 SimpleContainer Related Tags

Parents	Children
SimpleResult	SimpleText
	SimpleMenu
	SimpleForm
	SimpleTable
	SimpleAction
	SimpleBind

SimpleDisconnect

Disconnect's a connection oriented device like the Voice browser.

Table 8–21 SimpleDisconnect Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–22 SimpleDisconnect Related Tags

Parents	Children
SimpleCatch	none
SimpleCase	
SimpleTask	

SimpleDisplay

Supports all rendering characteristics of an SimpleBind (using SImpleTextItem). SimpleTextItem, a child SimpleDisplay, contains the actual render / display content. Useful in two cases i) Allows provides an audio (child of textitem), ii) Render the text for a MenuItem, when Bind is displayed a MenuItem (A Bind can be displayed as MenuItem if the SimpleMatch contains the SimpleMItem element and the SimpleBind occurs as child of SimpleMenu).

Table 8–23 SimpleDisplay Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Usage

```
SimpleDisplay as the rendering container of SimpleBind.
 <SimpleBind>
  </SimpleMatch>
  <SimpleTask>
  </SimpleTask>
  <SimpleDisplay>
    <SimpleTextItem deviceclass=".."><SimpleAudio .../></SimpleTextItem>
    <SimpleTextItem deviceclass="..">Hello welcome</SimpleTextItem>
    </SimpleDisplay>
```

Related Tags

Table 8–24 SimpleDisplay Related Tags

Parents	Children
SimpleBind	SimpleTextItem

SimpleDTMF

Specify a VoiceXML DTMF grammar. DTMF grammar can be used to indicate a syntax like 1 {San Francisco} | 2 {Wash. DC} | 3 {New York} etc. If the syntax information stored in a remote server, the "src" attribute can be used to specify the URI of the DTMF syntax resource / file.

Table 8–25 SimpleDTMF Tag

Name	Description	Value(s)	Default Value
src	URI to the resource file where the DTMF's / Grammars are stored	CDATA	OPTIONAL
type	The MIME type of the grammar. Represents the Grammar Format (applicable to both the remote URI Grammar file or inline Grammar text). There are different ways of representing a Grammar / DTMF format. Example: "application / x-jsgf",	CDATA	OPTIONAL
scope	VoiceXML scope. Can take "document" / "dialog" possible values. Default scope of grammar in Form / Menu / Text / Grammar / DTMF. If the scope is set to "document", then the grammar is active in the entire document. This allows for e.g. the user to submit a "Form" (in the SimpleResult) from inside a "SimpleMenu".	(document | dialog)	dialog OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). This tag is applicable for Voice devices only. Will not be supported on other devices even if specified.	CDATA	OPTIONAL

Related Tags

Table 8–26 SimpleDTMF Related Tags

Parents	Children
SimpleHref	[PCDATA]
SimpleAction	SimpleCache
SimpleForm	

Table 8–26 SimpleDTMF Related Tags

Parents	Children
SimpleFormItem	
SimpleFormSelect	
SimpleMatch	

SimpleEm

Displays the enclosed text(audio) with emphasis. Text enclosed usually displayed as italicized text.

Table 8–27 SimpleEm Tag

Name	Description	Value(s)	Default Value
level	Voice only attribute. Indicates Level of emphasis.	(strong | moderate | none | reduced)	moderate OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–28 SimpleEm Related Tags

Parents	Children
SimpleTextItem	[PCDATA]
SimpleSpan	SimpleBreak
SimpleUnderline	SimpleEm
SimpleImage	SimpleStrong
SimpleHref	SimpleSpan
SimpleStrong	SimpleUnderline
SimpleEm	SimpleAudio
SimpleAudio	SimpleSpeech
SimpleSpeech	SimpleImage
SimpleTitle	SimpleValue
SimpleHelp	

SimpleExit

Perform a application exit

Table 8–29 SimpleExit Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–30 SimpleExit Related Tags

Parents	Children
SimpleCatch	none
SimpleCase	
SimpleTask	

SimpleFinish

SimpleFinish indicates the Finish event. This is a tag is supported only in Voice. This can be any event that completes an user task. For example, reaching the end of the form field input on Voice devices.

Table 8–31 SimpleFinish Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–32 SimpleFinish Related Tags

Parents	Children
SimpleMatch	none

SimpleForm

This element is used for displaying one or more input fields. The fields are presented using the SimpleFormItem and SimpleFormSelect elements. Form has SimpleTitle as child, if specified will appear as the Title of the form.

Table 8–33 SimpleForm Tag

Name	Description	Value(s)	Default Value
layout	Control Layout of a form in small screen devices. Indicates if the form input fields should be displayed in a sequence of fields (cards) and should there be an enclosing page with all input field listed, allowing the user to select the field in an arbitrary fashion.	(linear | tabular)	linear OPTIONAL
scope	VoiceXML scope. Can take "document"/"dialog" possible values. Default scope of grammar in Form/Menu/Text/Grammar/DTMF. If the scope is set to "document", then the grammar is active in the entire document. This allows for e.g. the user to submit a "Form" (in the SimpleResult) from inside a "SimpleMenu".	(document | dialog)	dialog OPTIONAL
callbackurl	Wireless Module Support. Indicates the URL to return back if the current action leads the user into a different application (application implementing Wireless Module functionality).	CDATA	OPTIONAL
callbackparam	Wireless Module Support. Indicates the return parameters of the callbackurl. When Module returns the context back to the callee application, the callbackparam is passed back for the callee to construct its application state.	CDATA	OPTIONAL
callbacksecure	Indicates the mode of communication, when callback occurs, between Wireless server and the device. Setting callbacksecure="true" will enable a secure connect mode between Wireless and the device when the module performs a callback (to the callbackurl). If not specified, the connect mode will be based on the current request mode.	(true | false)	OPTIONAL

Table 8–33 SimpleForm Tag

Name	Description	Value(s)	Default Value
target	URI to navigate to when action is activated. This URL is always rewritten by the Server to point back to Wireless Server, except when mimetype attribute not "text/vnd.oracle.mobilexml". Also supports "callto:" for Phone call and "mailto:" for email support.	CDATA	OPTIONAL
mimetype	mime-type of target URI. Lets the Wireless server know the target resources mime-type. If the target mime-type is not text/vnd.oracle.mobilexml, the Wireless server will not rewrite the URL.	CDATA	text/vnd.oracle.mobilexml OPTIONAL
static_target	URI to navigate to when action is activated. This URL is never rewritten by server. If exists, this will override the "target" attribute. Also supports "callto:" for Phone call and "mailto:" for email support.	CDATA	OPTIONAL
secure	Indicates the mode of communication between Wireless server and the device. Setting secure="true" will enable a secure connect mode between Wireless and the device for the specified target. If not specified, the connect mode will be based on the current request mode. This DOES NOT indicate mode of connection between Wireless and the remote content source (the service), Wireless will connect to the remote service based on the protocol specified in the target attribute ("http" vs. "https")	(true | false)	OPTIONAL
fetchaudio	Voice only attribute. The URI of an audio clip to play while the "target" is being fetched.	CDATA	OPTIONAL

Table 8–33 SimpleForm Tag

Name	Description	Value(s)	Default Value
id	ID attribute of the element. Used for Navigation within a XML response (target="#ID")	xsd:ID	OPTIONAL
method	HTTP Method get or post	(get | post)	get OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–34 SimpleForm Related Tags

Parents	Children
SimpleContainer	SimpleTitle
	SimpleProperty
	SimpleTextItem
	SimpleTextField
	SimpleFormItem
	SimpleFormSelect
	SimpleCatch
	SimpleAction
	SimpleBind
	SimpleGrammar
	SimpleDTMF

SimpleFormItem

Specified if Input is mandatory. This attribute is supported only if the target device supports such an functionality. The application must always validate the field on the server side

Table 8–35 SimpleFormItem Tag

Name	Description	Value(s)	Default Value
mandatory	Specified if Input is mandatory. This attribute is supported only if the target device supports such an functionality. The application must always validate the field on the server side	(yes | no)	no OPTIONAL
maxlength	Max length of the field	Positive Int	OPTIONAL
type	Indicates the data type, like boolean, digits etc. For backward compatibility will also accept displaymode attribute values (text | textarea | password etc).	(none | audio | boolean | currency | date | digits | number | phone | time | transfer)	REQUIRED
format	WML / HDML format attribute. Supported only in WML / HDML devices	CDATA	OPTIONAL
value	default value ("defaultvalue" attribute also supported for backward compatibility).	CDATA	OPTIONAL
size	display size of the input field	Positive Int	OPTIONAL
name	Input Field name	CDATA	REQUIRED
displaymode	To Specify the display characteristics of the field like noecho (password), textarea etc.	(text | textarea | noecho | hidden)	text OPTIONAL
rows	Number of rows if displaymode is textArea.	Positive Int	OPTIONAL
cols	Number of rows if displaymode is textArea.	Positive Int	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Table 8–35 SimpleFormItem Tag

Name	Description	Value(s)	Default Value
modal	VoiceXML modal. If this is true (the default) all higher level speech and DTMF grammars are turned off while making the transcription. If false, the grammar is scoped to the form item / select.	(true | false)	true OPTIONAL
slot	VoiceXML slot. Slot exist part of a Voice Grammar syntax. The input's grammar slot values are assigned to the corresponding field item variables. This allows the user to say one sentence an fill-in more than form field.	CDATA	OPTIONAL
dest	VoiceXML dest. Valid only when the type="transfer". Specifies the phone number to transfer the call to.	CDATA	OPTIONAL
bridge	VoiceXML bridge. Valid only when the type="transfer". If "true" allows the original caller to resume the current session, once the transfer / third party call is complete.	(true | false)	false OPTIONAL
connecttimeout	VoiceXML connect timeout in milliseconds. Valid only when the type="transfer". The time to wait while trying to connect the call before returning the noanswer condition (Default is specific a Voice Gateway platform).	xsd:nonNegativeInteger	OPTIONAL
maxtime	VoiceXML maxtime in milli seconds. Valid only when the type="transfer" and brige="true". The time that the call is allowed to last, or 0 if it can last arbitrarily long. Default is 0.	xsd:nonNegativeInteger	OPTIONAL
beep	VoiceXML beep. If true, a tone is emitted just prior to transcription. Defaults to false. Used when type="audio".	(true | false)	true OPTIONAL
finalsilence	VoiceXML finalsilence milliseconds. The interval of silence that indicates end of speech.	xsd:nonNegativeInteger	OPTIONAL
enctype	VoiceXML enctype. The MIME encoding type of the submitted document. Used when type="audio" to indicate the format of the recording requested.	CDATA	OPTIONAL
dtmfterm	VoiceXML dtmfterm. If true, a DTMF keypress terminates transcription.	(true | false)	false OPTIONAL

Related Tags

Table 8–36 SimpleFormItem Related Tags

Parents	Children
SimpleForm	[PCDATA]
	SimpleTitle
	SimpleAction
	SimpleBind
	SimpleCatch
	SimpleProperty
	SimpleHelp
	SimpleGrammar
	SimpleDTMF

SimpleFormOption

Provides value for a formitem as a predefined list of values. Element is an item in a selectable menu. "FormOption" takes PCDATA and SimpleTextItem as child. SimpleTextItem is used to render the an Rich Text (useful when using Radio buttons and Checkboxes). Certain devices do not allow RichText as part of MenuItem, in such cases the text inside SimpleTextItem are collected and rendered. If SimpleTextItem does not exist, then the PCDATA is rendered. PCDATA is required, because in Voice the user select's the option by uttering the "PCDATA". "PCDATA",parsable character format, specifies default values for the form item.

Table 8–37 SimpleFormOption Tag

Name	Description	Value(s)	Default Value
selected	is this option selected by default (same semantics as HTML SELECTED)	(true | false)	false OPTIONAL
value	value of select variable when this is selected	CDATA	REQUIRED
dtmf	digit to be pressed on phone or DTMF tone. dtmf attribute just takes one value (a simplified form of voice SimpleDTMF tag). Will work on wap devices, if supported by the device.	CDATA	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–38 SimpleFormOption Related Tags

Parents	Children
SimpleFormSelect	[PCDATA]
SimpleOptGroup	SimpleTextItem

SimpleFormSelect

This element displays a select and option list. Can display option list, checkbox or radio box.

Table 8–39 SimpleFormSelect Tag

Name	Description	Value(s)	Default Value
displaymode	To Specify the display characteristics of the select like drop down, check box etc. Radio button if multiple is false and displaymode is checkbox. For backward compatibility should support the "type" attribute (values checkbox, radio).	(list | checkbox)	list OPTIONAL
multiple	Supports multiple options to be selected for the Select. Bot Supported in Voice	(true | false)	false OPTIONAL
name	Name of the select field	CDATA	REQUIRED
size	display size of the select list (if displaymode is list)	Positive Int	1 OPTIONAL
modal	VoiceXML modal. If this is true (the default) all higher level speech and DTMF grammars are turned off while making the transcription. If false, the grammar is scoped to the form item / select.	(true | false)	true OPTIONAL

Table 8–39 SimpleFormSelect Tag

Name	Description	Value(s)	Default Value
slot	VoiceXML slot. Slot exist part of a Voice Grammar syntax. The input's grammar slot values are assigned to the corresponding field item variables. This allows the user to say one sentence an fill-in more than form field.	CDATA	OPTIONAL
autoprompt	VoiceXML auto prompt. Tells the Voice browsers not to perform an auto prompt. Valid in menu's and formselect's. If set to false, the voice browser will not list the items in the menu / select. Typically set to false if need to use a audio file (listing all the menus, rather than using the TTS of the Voice gateway).	(true | false)	true OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–40 SimpleFormSelect Related Tags

Parents	Children
SimpleForm	SimpleTitle
	SimpleFormOption
	SimpleOptGroup
	SimpleCatch
	SimpleAction
	SimpleBind
	SimpleHelp
	SimpleGrammar
	SimpleDTMF

SimpleGo

Defines the "Go" task. Go is one the many possible tasks of a Bind operation (SimpleBind) and is defined as is child of SimpleTask. SimpleGo is an empty tag (no child tags).

Table 8–41 SimpleGo Tag

Name	Description	Value(s)	Default Value
callbackurl	Wireless Module Support. Indicates the URL to return back if the current action leads the user into a different application (application implementing Wireless Module functionary).	CDATA	OPTIONAL
callbackparam	Wireless Module Support. Indicates the return parameters of the callbackurl. When Module returns the context back to the callee application, the callbackparam is passed back for the callee to construct its application state.	CDATA	OPTIONAL
callbacksecure	Indicates the mode of communication, when callback occurs, between Wireless server and the device. Setting callbacksecure="true" will enable a secure connect mode between Wireless and the device when the module performs a callback (to the callbackurl). If not specified, the connect mode will be based on the current request mode.	(true | false)	OPTIONAL
target	URI to navigate to when action is activated. This URL is always rewritten by the Server to point back to Wireless Server, except when mimetype attribute not "text/vnd.oracle.mobilexml". Also supports "callto:" for Phone call and "mailto:" for email support.	CDATA	OPTIONAL
mimetype	mime-type of target URI. Lets the Wireless server know the target resources mime-type. If the target mime-type is not text/vnd.oracle.mobilexml, the Wireless server will not rewrite the URL.	CDATA	text/vnd.oracle.mobilexml OPTIONAL
static_target	URI to navigate to when action is activated. This URL is never rewritten by server. If exists, this will override the "target" attribute. Also supports "callto:" for Phone call and "mailto:" for email support.	CDATA	OPTIONAL

Table 8–41 SimpleGo Tag

Name	Description	Value(s)	Default Value
secure	Indicates the mode of communication between Wireless server and the device. Setting secure="true" will enable a secure connect mode between Wireless and the device for the specified target. If not specified, the connect mode will be based on the current request mode. This DOES NOT indicate mode of connection between Wireless and the remote content source (the service), Wireless will connect to the remote service based on the protocol specified in the target attribute ("http" vs. "https")	(true | false)	OPTIONAL
fetchaudio	Voice only attribute. The URI of an audio clip to play while the "target" is being fetched.	CDATA	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–42 SimpleGo Related Tags

Parents	Children
SimplePrev	SimpleCache
SimpleCatch	
SimpleCase	
SimpleTask	

SimpleGrammar

Provides the voice grammar for the enclosing item like SimpleMenuItem. For example: for SimpleMenuItem with enclosing text like "Oracle9*i*AS Wireless.", the Voice Engine would say "your options are Oracle9*i*AS Wireless.". Use SimpleGrammar for voice if you want to invoke this MenuItem when the user says "Oracle" | "Oracle9i" | "9i" | "Wireless"

Table 8–43 SimpleGrammar Tag

Name	Description	Value(s)	Default Value
src	URI to the resource file where the DTMF's / Grammar's are stored	CDATA	OPTIONAL
type	The MIME type of the grammar. Represents the Grammar Format (applicable to both the remote URI Grammar file or inline Grammar text). There are different ways of representing a Grammar / DTMF format. Example: "application / x-jsgf",	CDATA	OPTIONAL
scope	VoiceXML scope. Can take "document" / "dialog" possible values. Default scope of grammar in Form / Menu / Text / Grammar / DTMF. If the scope is set to "document", then the grammar is active in the entire document. This allows for e.g. the user to submit a "Form" (in the SimpleResult) from inside a "SimpleMenu".	(document | dialog)	dialog OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). This tag is applicable for Voice devices only. Will not be supported on other devices even if specified.	CDATA	OPTIONAL

Related Tags

Table 8–44 SimpleGrammar Related Tags

Parents	Children
SimpleHref	[PCDATA]
SimpleAction	SimpleCache

Table 8–44 SimpleGrammar Related Tags

Parents	Children
SimpleMenuItem	
SimpleForm	
SimpleFormItem	
SimpleFormSelect	
SimpleMatch	

SimpleHelp

Used to display Help text for a field. Used by SSD / PDA style devices to display help text for the FormItem / Select (In voice SimpleCatch "type="help" is used).

Table 8–45 SimpleHelp Tag

Name	Description	Value(s)	Default Value
color	color	CDATA	OPTIONAL
font	font	CDATA	OPTIONAL
size	size	CDATA	OPTIONAL
wrapmode	Text wrap mode.	(wrap | nowrap)	wrap OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–46 SimpleHelp Related Tags

Parents	Children
SimpleFormItem	[PCDATA]
SimpleFormSelect	SimpleHref
	SimpleBreak
	SimpleEm
	SimpleStrong
	SimpleImage

SimpleHref

Specifies a hyperlink.

Table 8–47 SimpleHref Tag

Name	Description	Value(s)	Default Value
callbackurl	Wireless Module Support. Indicates the URL to return back if the current action leads the user into a different application (application implementing Wireless Module functionary).	CDATA	OPTIONAL
callbackparam	Wireless Module Support. Indicates the return parameters of the callbackurl. When Module returns the context back to the callee application, the callbackparam is passed back for the callee to construct its application state.	CDATA	OPTIONAL
callbacksecure	Indicates the mode of communication, when callback occurs, between Wireless server and the device. Setting callbacksecure="true" will enable a secure connect mode between Wireless and the device when the module performs a callback (to the callbackurl). If not specified, the connect mode will be based on the current request mode.	(true | false)	OPTIONAL
target	URI to navigate to when action is activated. This URL is always rewritten by the Server to point back to Wireless Server, except when mimetype attribute not "text/vnd.oracle.mobilexml". Also supports "callto:" for Phone call and "mailto:" for email support.	CDATA	OPTIONAL
mimetype	mime-type of target URI. Lets the Wireless server know the target resources mime-type. If the target mime-type is not text/vnd.oracle.mobilexml, the Wireless server will not rewrite the URL.	CDATA	text/vnd.oracle.mobilexml OPTIONAL
static_target	URI to navigate to when action is activated. This URL is never rewritten by server. If exists, this will override the "target" attribute. Also supports "callto:" for Phone call and "mailto:" for email support.	CDATA	OPTIONAL

Table 8–47 SimpleHref Tag

Name	Description	Value(s)	Default Value
secure	Indicates the mode of communication between Wireless server and the device. Setting secure="true" will enable a secure connect mode between Wireless and the device for the specified target. If not specified, the connect mode will be based on the current request mode. This DOES NOT indicate mode of connection between Wireless and the remote content source (the service), Wireless will connect to the remote service based on the protocol specified in the target attribute ("http" vs. "https")	(true | false)	OPTIONAL
fetchaudio	Voice only attribute. The URI of an audio clip to play while the "target" is being fetched.	CDATA	OPTIONAL
label	Label for action button, displayed when action is bound to a button on a visual device	CDATA	OPTIONAL
dtmf	digit to be pressed on phone or DTMF tone. dtmf attribute just takes one value (a simplified form of voice SimpleDTMF tag). Will work on wap devices, if supported by the device.	CDATA	OPTIONAL
src	The URL to an image. Image from will be displayed. (In SimpleAction / Href this images needs to be used instead of the label.	CDATA	OPTIONAL

Table 8–47 SimpleHref Tag

Name	Description	Value(s)	Default Value
addImageExtension	Allows the server to use the right image format from a list of available formats. Based on the available images from the app (specified by the "available" attribute) and based on the device browser capability the server will pick the right image to be used. Example: if image is "oracle" and available is set to "jpg gif wbmp", server will use "oracle.wbmp" in WML (Phone.com) browser, "oracle.gif" for a HTML browser.	(true | false)	true OPTIONAL
available	Application can specify a list of available image formats, for example: available = "jpg gif g2.gif wbmp bmp" (g2.gif indicates a grayscale depth 2 image, for devices like Palm). This allows the server to use the correct image format supported by device (based on device browser properties).	CDATA	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–48 SimpleHref Related Tags

Parents	Children
SimpleTextItem	[PCDATA]
SimpleHelp	SimpleCache
	SimpleGrammar
	SimpleDTMF
	SimpleBreak
	SimpleEm

Table 8–48 SimpleHref Related Tags

Parents	Children
	SimpleStrong
	SimpleSpan
	SimpleUnderline
	SimpleAudio
	SimpleSpeech
	SimpleImage
	SimpleValue

SimpleImage

Table 8–49 SimpleImage Tag

Name	Description	Value(s)	Default Value
alt	alt string if Image not found	CDATA	OPTIONAL
border	Width of border	CDATA	OPTIONAL
width	Image width	CDATA	OPTIONAL
height	Image height	CDATA	OPTIONAL
vspace	Vertical space (from HTML)	CDATA	OPTIONAL
hspace	Horizontal space (from HTML)	CDATA	OPTIONAL
src	The URL to an image. Image from will be displayed. (In SimpleAction / Href this images needs to be used instead of the label.	CDATA	OPTIONAL
addImageExtension	Allows the server to use the right image format from a list of available formats. Based on the available images from the app (specified by the "available" attribute) and based on the device browser capability the server will pick the right image to be used. Example: if image is "oracle" and available is set to "jpg gif wbmp", server will use "oracle.wbmp" in WML (Phone.com) browser, "oracle.gif" for a HTML browser.	(true | false)	true OPTIONAL
available	Application can specify a list of available image formats for example: available = "jpg gif g2.gif wbmp bmp" (g2.gif indicates a grayscale depth 2 image, for devices like Palm). This allows the server to use the correct image format supported by device (based on device browser properties).	CDATA	OPTIONAL

Table 8–49 SimpleImage Tag

Name	Description	Value(s)	Default Value
valign	Vertical alignment	(top | middle | bottom)	top OPTIONAL
halign	Horizontal alignment	(left | center | right)	left OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–50 SimpleImage Related Tags

Parents	Children
SimpleTextItem	[PCDATA]
SimpleSpan	SimpleCache
SimpleUnderline	SimpleBreak
SimpleImage	SimpleEm
SimpleHref	SimpleStrong
SimpleStrong	SimpleSpan
SimpleEm	SimpleUnderline
SimpleAudio	SimpleAudio
SimpleSpeech	SimpleSpeech
SimpleTitle	SimpleImage
SimpleHelp	SimpleValue

SimpleKey

SimpleKey defines the device key for the Bind operation. SimpleKey, like SimpleAction has a type attribute that identified the Key on device for the Bind operation.

Table 8–51 SimpleKey Tag

Name	Description	Value(s)	Default Value
type	Defines the type of Binding in the target device. Can take any string value. Types primary, secondary are special values and map to the primary and secondary keys respectively. Also transformers will support "accept"|"soft1"|"option1"|"option2" (for backward compatibility).	(primary | secondary)	REQUIRED
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–52 SimpleKey Related Tags

Parents	Children
SimpleMatch	none

SimpleMatch

A Bind (SimpleBind) can be triggered by various actions like pressing a key, event or saying a key word (voice). Each of these actions are indicated by separate tags. SimpleMatch is the container tag for all such possible Bind Invocation tags.

Table 8–53 SimpleMatch Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–54 SimpleMatch Related Tags

Parents	Children
SimpleBind	SimpleFinish
	SimpleKey
	SimpleGrammar
	SimpleDTMF
	SimpleMItem
	SimpleEvent

SimpleMenu

This element represents a single menu with selectable links which are defined by the children SimpleMenuItem elements.

Table 8–55 SimpleMenu Tag

Name	Description	Value(s)	Default Value
scope	VoiceXML scope. Can take "document"/"dialog" possible values. Default scope of grammar in Form/Menu/Text/Grammar/DTMF. If the scope is set to "document", then the grammar is active in the entire document. This allows for e.g. the user to submit a "Form" (in the SimpleResult) from inside a "SimpleMenu".	(document | dialog)	dialog OPTIONAL
id	ID attribute of the element. Used for Navigation within a XML response (target="#ID")	xsd:ID	OPTIONAL
autoprompt	VoiceXML auto prompt. Tells the Voice browsers not to perform an auto prompt. Valid in menu's and formselect's. If set to false, the voice browser will not list the items in the menu/select. Typically set to false if need to use a audio file (listing all the menus, rather than using the TTS of the Voice gateway).	(true | false)	true OPTIONAL
wrapmode	Text wrap mode.	(wrap | nowrap)	wrap OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–56 SimpleMenu Related Tags

Parents	Children
SimpleContainer	SimpleTitle
	SimpleProperty
	SimpleCatch
	SimpleMenuItem
	SimpleBind
	SimpleAction

SimpleMenuItem

This element represents a single, selectable option in a menu defined by SimpleMenu.

Table 8–57 SimpleMenuItem Tag

Name	Description	Value(s)	Default Value
separator	If defined this adds visual separator after or before the menuitem (like the windows "Menu").	(before | after | none)	none OPTIONAL
callbackurl	Wireless Module Support. Indicates the URL to return back if the current action leads the user into a different application (application implementing Wireless Module functionary).	CDATA	OPTIONAL
callbackparam	Wireless Module Support. Indicates the return parameters of the callbackurl. When Module returns the context back to the callee application, the callbackparam is passed back for the callee to construct its application state.	CDATA	OPTIONAL
callbacksecure	Indicates the mode of communication, when callback occurs, between Wireless server and the device. Setting callbacksecure="true" will enable a secure connect mode between Wireless and the device when the module performs a callback (to the callbackurl). If not specified, the connect mode will be based on the current request mode.	(true | false)	OPTIONAL
target	URI to navigate to when action is activated. This URL is always rewritten by the Server to point back to Wireless Server, except when mimetype attribute not "text/vnd.oracle.mobilexml". Also supports "callto:" for Phone call and "mailto:" for email support.	CDATA	OPTIONAL
mimetype	mime-type of target URI. Lets the Wireless server know the target resources mime-type. If the target mime-type is not text/vnd.oracle.mobilexml, the Wireless server will not rewrite the URL.	CDATA	text/vnd.oracle.mobilexml OPTIONAL

Table 8–57 SimpleMenuItem Tag

Name	Description	Value(s)	Default Value
static_target	URI to navigate to when action is activated. This URL is never rewritten by server. If exists, this will override the "target" attribute. Also supports "callto:" for Phone call and "mailto:" for email support.	CDATA	OPTIONAL
secure	Indicates the mode of communication between Wireless server and the device. Setting secure="true" will enable a secure connect mode between Wireless and the device for the specified target. If not specified, the connect mode will be based on the current request mode. This DOES NOT indicate mode of connection between Wireless and the remote content source (the service), Wireless will connect to the remote service based on the protocol specified in the target attribute ("http" vs. "https")	(true | false)	OPTIONAL
fetchaudio	Voice only attribute. The URI of an audio clip to play while the "target" is being fetched.	CDATA	OPTIONAL
label	Label for action button, displayed when action is bound to a button on a visual device	CDATA	OPTIONAL
dtmf	digit to be pressed on phone or DTMF tone. dtmf attribute just takes one value (a simplified form of voice SimpleDTMF tag). Will work on wap devices, if supported by the device.	CDATA	OPTIONAL
wrapmode	Text wrap mode.	(wrap | nowrap)	wrap OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–58 SimpleMenuItem Related Tags

Parents	Children
SimpleMenu	[PCDATA]
	SimpleCache
	SimpleTextItem
	SimpleGrammar

SimpleMeta

Defines all WML / HDML / HTML meta tags (pass through)

Table 8–59 SimpleMeta Tag

Name	Description	Value(s)	Default Value
content	The content of the emulated HTTP header or associated content of Meta NAME	CDATA	REQUIRED
http-equiv	The equivalent HTTP header you are emulating	CDATA	REQUIRED
name	a descriptive name of the meta attribute	CDATA	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–60 SimpleMeta Related Tags

Parents	Children
SimpleResult	none

SimpleMItem

Empty tag to indicate the Bind needs to rendered as a MenuItem. This is allowed only when SimpleBind is a child of SimpleMenu. Use SimpleTextItem, as a child of SimpleDisplay, to display for the actual text of a menu item text. Defines all WML / HDML / HTML meta tags (pass through)

Table 8–61 SimpleMItem Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–62 SimpleMItem Related Tags

Parents	Children
SimpleMatch	none

SimpleName

Identifies Client side form field names. Used to specify a list of client side form fields, that need to cleared. Useful in voice, as clearing of form fields allows for reprompt by the VoiceXML browser.

Table 8–63 SimpleName Tag

Name	Description	Value(s)	Default Value
name	Name of Client side form field	CDATA	REQUIRED
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–64 SimpleName Related Tags

Parents	Children
SimpleClear	none
SimpleSubmit	

SimpleOptGroup

Group SimpleFormOptions into a hierarchy. To support Small screen devices, where long lists of options cannot not deliver good UIs. On devices where optgroup is not supported the display strings of options, inside the optgroups, are concatenated with label defined in the optgroup.

Table 8–65 SimpleOptGroup Tag

Name	Description	Value(s)	Default Value
label	For platforms that support hierarchical option lists, the label is displayed when navigating non-leaf nodes	CDATA	REQUIRED
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–66 SimpleOptGroup Related Tags

Parents	Children
SimpleFormSelect	[PCDATA]
	SimpleFormOption

SimplePhone

A Bind (SimpleBind) can be triggered by any event and also these can be device specific events. The SimpleEvent element describes the possible events that would trigger the Bind action. This Element allows you to take advantage of device specific event handlers and define actions that can be triggered on such events. The attribute "type" identifies the device specific events. For Voice applications you can use events like "noinput", "cancel" etc. For WML it can be events like "onenterforward", "onpick" etc.

Table 8–67 SimplePhone Tag

Name	Description	Value(s)	Default Value
count	Applicable to Voice events only. The occurrence of the event (default is 1). The count allows you to handle different occurrences of the same event differently for example. If need to give extra help messages if the user says "help" twice for the same form item. The form / formitem / menu etc (where ever SimpleCatch can occur) maintains a counter for each event that occurs while it is being visited, these counters are reset each time the form is re-entered.	Positive Int	1 OPTIONAL
type	Predefined device level events. Possible values for voice include for cancel, error, exit, help, noinput, nomatch, telephone.disconnect etc. Possible values for WML devices include "onpick", "onenterforward" etc.	CDATA	REQUIRED
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–68 SimplePhone Related Tags

Parents	Children
SimpleMatch	none

SimplePrev

Tag for the "PREV" (previous) functionality. Has SimpleGo as child and the target of the SimpleGo is the destination URL if "PREV" is not supported natively by the browser.

Table 8–69 SimplePrev Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–70 SimplePrev Related Tags

Parents	Children
SimpleCatch	SimpleGo
SimpleCase	
SimpleTask	

SimpleProperty

Set VoiceXML engine properties

Table 8–71 SimpleProperty Tag

Name	Description	Value(s)	Default Value
name	The name of a property.	CDATA	REQUIRED
value	The name of a property.	CDATA	REQUIRED
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–72 SimpleProperty Related Tags

Parents	Children
SimpleForm	none
SimpleFormItem	
SimpleResult	
SimpleMenu	

SimpleRefresh

Perform a refresh of the device if supported by the device

Table 8–73 SimpleRefresh Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–74 SimpleProperty Related Tags

Parents	Children
SimpleCatch	none
SimpleCase	
SimpleTask	

SimpleReprompt

This task will reprompt the user for the field input. Valid in Voice apps only and used for reprompting the form fields/inputs.

Table 8–75 SimpleReprompt Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–76 SimpleReprompt Related Tags

Parents	Children
SimpleCatch	none
SimpleCase	
SimpleTask	

SimpleResult

The root tag of Wireless XML. SimpleResult encloses the complete response for a request

Table 8–77 SimpleResult Tag

Name	Description	Value(s)	Default Value
application	VoiceXML application. Attribute used in voice (VoiceXML). This is an URL, which points to "root" document for the VoiceXML generated.	CDATA	OPTIONAL
bgcolor	Sets the Background color in supported devices	CDATA	OPTIONAL
lang	language of this document. Used for Voice, indicates the language of the XML document	CDATA	OPTIONAL

Related Tags

Table 8–78 SimpleResult Related Tags

Parents	Children
none	SimpleMeta
	SimpleCatch
	SimpleProperty
	SimpleCache
	SimpleAction
	SimpleBind
	SimpleContainer
	SimpleTimer

SimpleRow

Defines row of a table.

Table 8–79 SimpleRow Tag

Name	Description	Value(s)	Default Value
bgcolor	Sets the Background color in supported devices	CDATA	OPTIONAL
bordercolor	from HTML table spec	CDATA	OPTIONAL
valign	Vertical alignment	(top | middle | bottom)	top OPTIONAL
halign	Horizontal alignment	(left | center | right)	left OPTIONAL
wrapmode	Text wrap mode	(wrap | nowrap)	wrap OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–80 SimpleRow Related Tags

Parents	Children
SimpleTableBody	SimpleCol

SimpleSpan

Element to control Style of Text. Control for font, color and size of text.

Table 8–81 SimpleSpan Tag

Name	Description	Value(s)	Default Value
color	Color	CDATA	OPTIONAL
font	Font	CDATA	OPTIONAL
size	Font size	CDATA	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–82 SimpleSpan Related Tags

Parents	Children
SimpleTextItem	[PCDATA]
SimpleSpan	SimpleBreak
SimpleUnderline	SimpleEm
SimpleImage	SimpleStrong
SimpleHref	SimpleSpan
SimpleStrong	SimpleUnderline
SimpleEm	SimpleAudio
SimpleAudio	SimpleSpeech
SimpleSpeech	SimpleImage
SimpleTitle	SimpleValue

SimpleSpeech

Control prosody, class, and other VoiceXML text-to-speech engine parameters.

Table 8–83 SimpleSpeech Tag

Name	Description	Value(s)	Default Value
class	VoiceXML 'sayas' class. Allows the Voice browser to say something like "6505067000" as phone number, when class="phone"(rather than saying this as number which would 6 million ...).	(phone | date | digits | literal | currency | number | time	OPTIONAL
phon	VoiceXML 'sayas' phonetics. The representation of the Unicode International Phonetic Alphabet (IPA) characters that are to be spoken instead of the contained text.	CDATA	OPTIONAL
pitch	VoiceXML prosody pitch. numeric attribute that sets the baseline pitch in Hertz. Values can be "n" (set volume to n) or +n or -n. Also can be +n% , -n% or reset.	CDATA	OPTIONAL
range	VoiceXML prosody range. numeric attribute that sets the pitch range in Hertz. Values can be "n" (set volume to n) or +n or -n. Also can be +n% , -n% or reset. The pitch range represents the amount of variation in pitch above the baseline.	CDATA	OPTIONAL
rate	numeric attribute that sets the speaking rate in words per minute. Value Can be an exact number like "150" (sets the speaking rate of 150 words per minute) or can be +n (or -n) (increase or decrease the rate by n from the current level). Also can be +n% , -n% or reset	CDATA	OPTIONAL

Table 8–83 SimpleSpeech Tag

Name	Description	Value(s)	Default Value
sub	VoiceXML 'sayas' sub. Defines substitute text to be spoken instead of the contained text.	CDATA	OPTIONAL
vol	VoiceXML prosody volume. Numeric attribute that sets the output volume on a scale of 0.0 to 1.0 where 0.0 is silence and 1.0 is maximum loudness. Values can be "n" (set volume to n) or +n or -n. Also can be +n% , -n% or reset	CDATA	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). This tag is applicable for Voice devices only. Will not be supported on other devices even if specified.	CDATA	OPTIONAL

Related Tags

Table 8–84 SimpleSpeech Related Tags

Parents	Children
SimpleTextItem	[PCDATA]
SimpleSpan	SimpleBreak
SimpleUnderline	SimpleEm
SimpleImage	SimpleStrong
SimpleHref	SimpleSpan
SimpleStrong	SimpleUnderline
SimpleEm	SimpleAudio
SimpleAudio	SimpleSpeech
SimpleSpeech	SimpleImage
SimpleTitle	SimpleValue

SimpleStrong

Displays enclosed text in a stronger representation, usually bold

Table 8–85 SimpleStrong Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–86 SimpleStrong Related Tags

Parents	Children
SimpleTextItem	[PCDATA]
SimpleSpan	SimpleBreak
SimpleUnderline	SimpleEm
SimpleImage	SimpleStrong
SimpleHref	SimpleSpan
SimpleStrong	SimpleUnderline
SimpleEm	SimpleAudio
SimpleAudio	SimpleSpeech
SimpleSpeech	SimpleImage
SimpleTitle	SimpleValue
SimpleHelp	

SimpleSubmit

Defines the Submit task of a Bind. SimpleSubmit is child of SimpleTask. SimpleSubmit bind performs a submit action. You may provide a list of Form item names that has to be submitted. If provide a name list is provided then only those form item will be submitted. An Empty SimpleSubmit will Submit all the form items.

Table 8–87 SimpleSubmit Tag

Name	Description	Value(s)	Default Value
name	Name of the Submit button / action (just like HTML). The Submit "name" and "Value" will be submitted back to the app as parameters.	CDATA	REQUIRED
value	Value of the Submit button / action (just like HTML). The Submit "name" and "value" will be submitted back to the app as parameters.	CDATA	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL
method	HTTP Method get or post	(get | post)	get OPTIONAL
callbackurl	Wireless Module Support. Indicates the URL to return back if the current action leads the user into a different application (application implementing Wireless Module functionary).	CDATA	OPTIONAL
callbackparam	Wireless Module Support. Indicates the return parameters of the callbackurl. When Module returns the context back to the callee application, the callbackparam is passed back for the callee to construct its application state.	CDATA	OPTIONAL

Table 8–87 SimpleSubmit Tag

Name	Description	Value(s)	Default Value
callbacksecure	Indicates the mode of communication, when callback occurs, between Wireless server and the device. Setting callbacksecure="true" will enable a secure connect mode between Wireless and the device when the module performs a callback (to the callbackurl). If not specified, the connect mode will be based on the current request mode.	(true | false)	OPTIONAL
target	URI to navigate to when action is activated. This URL is always rewritten by the Server to point back to Wireless Server, except when mimetype attribute not "text/vnd.oracle.mobilexml". Also supports "callto:" for Phone call and "mailto:" for email support.	CDATA	OPTIONAL
mimetype	mime-type of target URI. Lets the Wireless server know the target resources mime-type. If the target mime-type is not text/vnd.oracle.mobilexml, the Wireless server will not rewrite the URL.	CDATA	text/vnd.oracle.mobilexml OPTIONAL
static_target	URI to navigate to when action is activated. This URL is never rewritten by server. If exists, this will override the "target" attribute. Also supports "callto:" for Phone call and "mailto:" for email support.	CDATA	OPTIONAL
secure	Indicates the mode of communication between Wireless server and the device. Setting secure="true" will enable a secure connect mode between Wireless and the device for the specified target. If not specified, the connect mode will be based on the current request mode. This DOES NOT indicate mode of connection between Wireless and the remote content source (the service), Wireless will connect to the remote service based on the protocol specified in the target attribute ("http" vs. "https")	(true | false)	OPTIONAL
fetchaudio	Voice only attribute. The URI of an audio clip to play while the "target" is being fetched.	CDATA	OPTIONAL

Related Tags

Table 8–88 SimpleSubmit Related Tags

Parents	Children
SimpleCatch	SimpleName
SimpleCase	
SimpleTask	

SimpleSwitch

Tag to write switch statements on form field name / value. Allows to compare the Value of the form field input on the client side and can branch to perform different Tasks.

Table 8–89 SimpleSwitch Tag

Name	Description	Value(s)	Default Value
name	Name of the form field the switch is based on.	CDATA	REQUIRED
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–90 SimpleSwitch Related Tags

Parents	Children
SimpleTask	SimpleCase

SimpleTable

Table 8–91 SimpleTable Tag

Name	Description	Value(s)	Default Value
separator	Used when table is not supported by the target device. If defined add a separator between column values where table cannot be supported.	CDATA	none OPTIONAL
id	ID attribute of the element. Used for Navigation within a XML response (target="#ID")	xsd:ID	OPTIONAL
bgcolor	background color	CDATA	OPTIONAL
border	Width of Border.	CDATA	OPTIONAL
bordercolor	Table bordercolor	CDATA	OPTIONAL
cellpadding	Cellpadding. As in HTML table	CDATA	OPTIONAL
cellspacing	Cellspacing. As in HTML table	CDATA	OPTIONAL
width	table Width	CDATA	OPTIONAL
height	table Height	CDATA	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–92 SimpleTable Related Tags

Parents	Children
SimpleContainer	SimpleTableHeader
	SimpleTableBody

SimpleTableBody

Table 8–93 SimpleTableBody Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–94 SimpleTableBody Related Tags

Parents	Children
SimpleTable	SimpleRow

SimpleTableHeader

Table 8–95 SimpleTableHeader Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–96 SimpleTableHeader Related Tags

Parents	Children
SimpleTable	SimpleCol

SimpleTask

Container tag for all task items of a Bind (SimpleBind). Tag encloses all the possible tasks like go, submit, exit etc. Task also includes TextItem as a child, this allows rendering of an audio or text (speech) before performing an action (useful in voice applications)

Table 8–97 SimpleTask Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–98 SimpleTask Related Tags

Parents	Children
SimpleBind	SimpleSwitch
	SimpleGo
	SimpleTextItem
	SimpleRefresh
	SimpleClear
	SimpleReprompt
	SimpleExit
	SimpleDisconnect
	SimplePrev
	SimpleSubmit

SimpleText

Container for block of Texts (SimpleTextItem's)

Table 8–99 SimpleText Tag

Name	Description	Value(s)	Default Value
wait	VoiceXML Wait. Tells The voice browser if a wait has to happen before proceeding to the next construct in the SimpleResult.	(true | false)	true OPTIONAL
wrapmode	Text wrap mode.	(wrap | nowrap)	wrap OPTIONAL
scope	VoiceXML scope. Can take "document"/"dialog" possible values. Default scope of grammar in Form/Menu/Text/Grammar/DTMF. If the scope is set to "document", then the grammar is active in the entire document. This allows for e.g. the user to submit a "Form" (in the SimpleResult) from inside a "SimpleMenu".	(document | dialog)	dialog OPTIONAL
id	ID attribute of the element. Used for Navigation within a XML response (target="#ID")	xsd:ID	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–100 SimpleText Related Tags

Parents	Children
SimpleContainer	SimpleTitle
	SimpleCatch
	SimpleTextItem

Table 8–100 SimpleText Related Tags

Parents	Children
	SimpleAction
	SimpleBind

SimpleTextField

Used to display non-editable field inside a form. For example, changing an password, where the userid is an non-editable field.

Table 8–101 SimpleTextField Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–102 SimpleTextField Related Tags

Parents	Children
SimpleForm	SimpleTitle
	SimpleTextItem

SimpleTextItem

This element contains one block of plain text, typically a single paragraph.

Table 8–103 SimpleTextItem Tag

Name	Description	Value(s)	Default Value
timeout	VoiceXML timeout. The the interval of silence before the next construct is played	xsd:nonNegativeInteger	OPTIONAL
color	color	CDATA	OPTIONAL
font	font	CDATA	OPTIONAL
size	font size	CDATA	OPTIONAL
wrapmode	Text wrap mode	(wrap | nowrap)	wrap OPTIONAL
bargein	VoiceXML bargein. Control whether a user can interrupt a when the text is being read by the VoiceXML browser.	(true | false)	true OPTIONAL
count	VoiceXML count. A number that allows you to emit different prompts if the user is doing something repeatedly. If omitted, it defaults to "1".	Positive Int	1 OPTIONL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–104 SimpleTextItem Related Tags

Parents	Children
SimpleText	[PCDATA]
SimpleForm	SimpleHref
SimpleTextField	SimpleBreak

Table 8–104 SimpleTextItem Related Tags

Parents	Children
SimpleFormOption	SimpleEm
SimpleCol	SimpleStrong
SimpleAction	SimpleSpan
SimpleDisplay	SimpleUnderline
SimpleMenuItem	SimpleAudio
SimpleCatch	SimpleSpeech
SimpleCase	SimpleImage
SimpleTask	SimpleValue

SimpleTimer

Invokes a "goto" target task after a specified delay time.

Table 8–105 SimpleTimer Tag

Name	Description	Value(s)	Default Value
timer	Invokes a "goto" target task after a specified delay time. Time in milliseconds	xsd:nonNegativeInteger	REQUIRED
target	URI to navigate to when action is activated. This URL is always rewritten by the Server to point back to Wireless Server, except when mimetype attribute not "text/vnd.oracle.mobilexml". Also supports "callto:" for Phone call and "mailto:" for email support.	CDATA	OPTIONAL
mimetype	mime-type of target URI. Lets the Wireless server know the target resources mime-type. If the target mime-type is not text/vnd.oracle.mobilexml, the Wireless server will not rewrite the URL.	CDATA	text/vnd.oracle.mobilexml OPTIONAL
static_target	URI to navigate to when action is activated. This URL is never rewritten by server. If exists, this will override the "target" attribute. Also supports "callto:" for Phone call and "mailto:" for email support.	CDATA	OPTIONAL

Table 8–105 SimpleTimer Tag

Name	Description	Value(s)	Default Value
secure	Indicates the mode of communication between Wireless server and the device. Setting secure="true" will enable a secure connect mode between Wireless and the device for the specified target. If not specified, the connect mode will be based on the current request mode. This DOES NOT indicate mode of connection between Wireless and the remote content source (the service), Wireless will connect to the remote service based on the protocol specified in the target attribute ("http" vs. "https")	(true | false)	OPTIONAL
fetchaudio	Voice only attribute. The URI of an audio clip to play while the "target" is being fetched.	CDATA	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–106 SimpleTimer Related Tags

Parents	Children
SimpleResult	none

SimpleTitle

Title element for form Field and Menu container (SimpleMenu)

Table 8–107 SimpleTitle Tag

Name	Description	Value(s)	Default Value
timeout	VoiceXML timeout. The the interval of silence allowed while waiting for user input in a form input (after prompting the user for input).	xsd:nonNegativeInteger	OPTIONAL
color	Color	CDATA	OPTIONAL
font	Font	CDATA	OPTIONAL
size	Font size	CDATA	OPTIONAL
bargein	VoiceXML bargein. Control whether a user can interrupt a when the text is being read by the VoiceXML browser.	(true | false)	true OPTIONAL
count	VoiceXML count. A number that allows you to emit different prompts if the user is doing something repeatedly. If omitted, it defaults to "1".	Positive Int	1 OPTIONAL
wrapmode	Text wrap mode.	(wrap | nowrap)	wrap OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–108 SimpleTitle Related Tags

Parents	Children
SimpleText	[PCDATA]
SimpleMenu	SimpleBreak

Table 8–108 SimpleTitle Related Tags

Parents	Children
SimpleForm	SimpleEm
SimpleTextField	SimpleStrong
SimpleFormItem	SimpleSpan
SimpleFormSelect	SimpleUnderline
	SimpleAudio
	SimpleSpeech
	SimpleImage
	SimpleValue

SimpleUnderline

Underline a text.

Table 8–109 SimpleUnderline Tag

Name	Description	Value(s)	Default Value
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–110 SimpleUnderline Related Tags

Parents	Children
SimpleTextItem	[PCDATA]
SimpleSpan	SimpleBreak
SimpleUnderline	SimpleEm
SimpleImage	SimpleStrong
SimpleHref	SimpleSpan
SimpleStrong	SimpleUnderline
SimpleEm	SimpleAudio
SimpleAudio	SimpleSpeech
SimpleSpeech	SimpleImage
SimpleTitle	SimpleValue

SimpleValue

Substitute the value of the client side form field variable, just like a macro. Possible with WML, VoiceXML etc. Can be used to provide a client side confirmation display / screen like "you entered 5, do you want continue" (Where the value 5 is the value of a form item).

Table 8–111 SimpleValue Tag

Name	Description	Value(s)	Default Value
audiobase	VoiceXML base from value element	CDATA	OPTIONAL
class	VoiceXML 'class' from value element. Can take any value on the enumerated list (or can be any string).	(none | audio | boolean | currency | date | digits | number | phone | time | transfer)	OPTIONAL
mode	VoiceXML mode. The type of rendering: tts (the default), or recorded. Can use the audiobase attribute to specify base directory of the audio files	CDATA	OPTIONAL
name	Name of the client variable to substitute	CDATA	OPTIONAL
deviceclass	This tag is interpreted only for the specified deviceclass (conditional transform). The server will transform this element only for certain devices (all devices that belong to the specified deviceclass). Values can be any combination of following "pdabrowser", "pcbrowser", "voice", "microbrowser", "micromessenger", "messenger". If not specified, the tag is interpreted for all devices.	CDATA	OPTIONAL

Related Tags

Table 8–112 SimpleValue Related Tags

Parents	Children
SimpleTextItem	none
SimpleSpan	
SimpleUnderline	
SimpleImage	

Table 8–112 SimpleValue Related Tags

Parents	Children
SimpleHref	
SimpleStrong	
SimpleEm	
SimpleAudio	
SimpleSpeech	
SimpleTitle	

8.2 Using Mobile Context Information in XML

Mobile Contexts are equivalent to scriptlets in many scripting languages like JSP, ASP etc. Mobile Scripting is primarily a context variable substitution. In other words, Mobile Contexts are placeholder for properties substituted by Oracle9*i*AS Wireless core at the runtime. Though, embedded Mobile Context are not literal to Oracle9*i*AS Wireless runtime, nonetheless, they do not violate rules of XML document. Oracle9*i*AS Wireless predefines a set of Mobile Contexts for application developers to use.

Oracle9*i*AS Wireless also sends all the Mobile Context information as HTTP Headers while invoking a request. It allows application developers to retrieve the Mobile Context information as HTTP Headers. Mobile Contexts as HTTP may be used to make any application-level decisions, or may be used for generating responses while embedded Mobile Contexts may only be used for generating responses.

Mobile Contexts are primarily divided into following four categories.

- User Context
- User Location Context
- Service Context
- Module Context

Table 8–113 User Context

Variable Name	DataType	Description
user.name	String	Login name of the User. If anonymous user, this should not be set.

Table 8–113 User Context

Variable Name	DataType	Description
user.displayname	String	Display name of the User.

Table 8–114 User Location Context

Variable Name	DataType	Description
user.location.addressline1	String	Address line1 of the location
user.location.addressline2	String	Address line2 of the location
user.location.companyname	String	Company name of the address
user.location.addresslastline	String	Address line3 of the location
user.location.block	String	Location Block
user.location.city	String	Location City
user.location.county	String	Location county
user.location.state	String	Location state
user.location.postalcode	String	Location zip / postal code
user.location.postalcodeext	String	Extended zip / postal code
user.location.country	String	Country
user.location.type	String	Values are "profile" / "auto"

Table 8–115 Service Context

Variable Name	DataType	Description
service.home.url	String	URL to the Home Page of the current service.
home.url	String	URL to the User's Wireless home Page
service.parent.Url	String	URL to folder container

Table 8–116 Module Context

Variable Name	DataType	Description
module.callback.url	String	The callback URL for Module return statement.

Table 8–116 Module Context

Variable Name	DataType	Description
module.callback.label	String	Display Label for the Module, calling back the caller.

8.3 Using Mobile Context Information from HTTP Headers

Table 8–117 User Context

Header Name	DataType	Description
X-Oracle-User.Locale	String	User Locale Information

Table 8–118 User Locale

Header Name	DataType	Description
X-Oracle-User.name	String	Login name of the User. If anonymous user, this should not be set. *(Value Encoded based on InputEncoding Setting. See Section on Encoding)*
X-Oracle-User.DisplayName	String	Display name of the User. *(Value Encoded based on InputEncoding Setting. See Section on Encoding)*
X-Oracle-User.userkind	String	Indicates if user is "anonymous", "virtual" (implicit Identity) or "registered".
X-Oracle-User.authkind	String	Indicates current session's auth mode, values are "unauthenticated", "weak" (weak authentication, implicit identity) or "authenticated"

Table 8–119 User Location Context

Header Name	DataType	Description
X-Oracle-User.Location.AddressLine1	String	Address line1 of the location. (Value Encoded based on InputEncoding Setting. See Section on Encoding)
X-Oracle-User.Location.AddressLine2	String	Address line2 of the location. (Value Encoded based on InputEncoding Setting. See Section on Encoding)
X-Oracle-User.Location.Companyname	String	Company name at the address.(Value Encoded based on InputEncoding Setting. See Section on Encoding)
X-Oracle-User.Location.AddressLastLine	String	Address line3 of the location. (Value Encoded based on InputEncoding Setting. See Section on Encoding)

Table 8–119 User Location Context

Header Name	DataType	Description
X-Oracle-User.Location.Block	String	Location Block. (Value Encoded based on InputEncoding Setting. See Section on Encoding)
X-Oracle-User.Location.City	String	Location City. (Value Encoded based on InputEncoding Setting. See Section on Encoding)
X-Oracle-User.Location.County	String	Location county. (Value Encoded based on InputEncoding Setting. See Section on Encoding)
X-Oracle-User.Location.State	String	Location state. (Value Encoded based on InputEncoding Setting. See Section on Encoding)
X-Oracle-User.Location.PostalCode	String	Location zip / postal code. (Value Encoded based on InputEncoding Setting. See Section on Encoding)
X-Oracle-User.Location.PostalCodeExt	String	Extended zip / postal code. (Value Encoded based on InputEncoding Setting. See Section on Encoding)
X-Oracle-User.Location.Country	String	Country. (Value Encoded based on InputEncoding Setting. See Section on Encoding)
X-Oracle-User.Location.Type	String	Values are "profile" or "auto"
X-Oracle-User.Location.X	String	X Cord of the location (float)
X-Oracle-User.Location.Y	String	Y Cord of the location (float)

Table 8–120 Service Context

Header Name	DataType	Description
X-Oracle-Service.Home.Url	String	URL to the Home deck of the current service.
X-Oracle-Service.Parent.Url	String	URL to folder container.
X-Oracle-Home.Url	String	URL to the User's Wireless home deck.

Table 8–121 Module Context

Header Name	DataType	Description
X-Oracle-Module.CallBack.Url	String	The callback URL for Module return statement.
X-Oracle-Module.CallBack.Label	String	Display Label for the Module, calling back the caller.

8.3.1 Encoding and Escaping Locale String from Request

Headers are encoded in ISO8859-1 character set according to HTTP 1.1 specification.

All request parameters and certain Headers as specified above (location, user etc.) are encoded as described by encoding parameter of service definition. Further the values are URL encoded as per HTTP 1.1 specification.

Application may retrieve these requests values by performing the following steps.

1. Use URL decoding to undo base64 encoding.
2. Construct the new string using service specific encoding.

For example a typical jsp scriptlet for Big5 (traditional Chinese) may look as follows for Java Programming Language.

```
<%
// let the encoding of service be Big5
String userName = request.getHeader("X-Oracle-User.DisplayName");
userName = java.net.URLDecoder.decode(userName);
userName = new String(userName.getBytes(), “Big5”);
%>
```

8.3.1.1 User Location Context

Table 8–122 User Location Context

Variable Name	DataType	Description
user.location.addressline1	String	Address line1 of the location
user.location.addressline2	String	Address line2 of the location
user.location.companyname	String	Company name of the address
user.location.addresslastline	String	Address line3 of the location
user.location.block	String	Location Block
user.location.city	String	Location City
user.location.county	String	Location county
user.location.state	String	Location state
user.location.postalcode	String	Location zip / postalcode
user.location.postalcodeext	String	Extended zip / postal code
user.location.country	String	Country
user.location.type	String	Values are "profile" / "auto"

8.3.1.2 Service Context

Table 8–123 Service Context

Variable Name	DataType	Description
service.home.url	String	URL to the Home Page of the current service.
home.url	String	URL to the User's Wireless home Page
service.parent.Url	String	URL to folder container.

8.3.1.3 Module Context

Table 8–124 Module Context

Variable Name	DataType	Description
module.callback.url	String	The callback URL for Module return statement.
module.callback.label	String	Display Label for the Module, calling back the caller.

For example, the following two applications can be used to greet the user.

8.3.1.4 HelloUserMobileScript.xml

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
 <SimpleContainer>
  <SimpleText>
   <SimpleTextItem>
    <SimpleStrong>Hello %value user.displayname%</SimpleStrong>
   </SimpleTextItem>
  </SimpleText>
 </SimpleContainer>
</SimpleResult>
```

8.3.1.5 HelloUserMobileScriptHTTP.jsp

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<%
String userName = request.getHeader("X-Oracle-User.DisplayName");
userName = (userName == null) ? request.getHeader("X-Oracle-User.name") :
userName;
userName = (userName == null) ? "Visitor" : userName;
userName = java.net.URLDecoder.decode(userName);
%>
<SimpleResult>
 <SimpleContainer>
  <SimpleText>
   <SimpleTextItem halign="center">Hello <%=userName%></SimpleTextItem>
  </SimpleText>
 </SimpleContainer>
</SimpleResult>
```

Figure 8–1 Output of HelloUserMobileScript.xml and HelloUserMobileScriptHTTP.jsp

Part III

Oracle9*i*AS Wireless Platform and Services

Part III contains information about the Oracle9*i*AS Wireless platform and services.

9

Mobile Service Developer's Tools

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

- Section 9.1, "Mobile Studio"
- Section 9.2, "Oracle9iAS Wireless SDK"
- Section 9.3, "Overview of JDeveloper with Oracle9iAS Wireless"
- Section 9.4, "Third-party Mobile Simulators"
- Section 9.5, "Deploying Your Applications"

9.1 Mobile Studio

9.1.1 In-house Mobile Studio

Mobile Studio is an online environment for quickly building, testing and deploying wireless applications. It lets any developer quickly develop mobile applications that are immediately accessible from all devices.

As a developer, you do not need to download or install any software to start using the Studio; provides a completely web-based development and testing environment. To access Mobile Studio on your Oracle9*i*AS Wireless instance, go to: http://*oracle9iasw-host:port*/studio.

9.1.1.1 Register with Mobile Studio

As a developer, you must register with your instance of Studio to access the Studio web site. To register, click on the register button on the Studio home page and provide the required details to register. Once you register, Studio provides you with a personal application area to test your applications and also provides links to sample applications.

9.1.1.2 Develop HelloWorld Application

In this walkthrough you will create a HelloWorld mobile application.

1. First you will create simple static page that outputs Oracle9*i*AS Wireless XML when accessed through a web server.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
    <SimpleContainer>
        <SimpleText>
            <SimpleTitle>HelloWorld Page</SimpleTitle>
            <SimpleTextItem>Hello World</SimpleTextItem>
        </SimpleText>
    </SimpleContainer>
</SimpleResult>
```

2. Host the above *helloworld.xml* sample page on you web server.
3. Log into the Studio and select the CreateNew application button.
4. In the Create Application page fill in the following:

   Name: MyHelloWorld

   URL: URL to the helloworls.xml file (such as http://server:port/….)

   Deviceclass: Select “all devices”

   Description: My First Studio Sample

5. Click Create.

9.1.1.3 Test and Debug HelloWorld Application

In this walkthrough you will test and debug the HelloWorld application created in the previous walkthrough.

1. Start the Mobile Simulator.
2. Enter the following URL in the Go window:
   http://9iASWEServer.domain/ptg/rm
   This is the URL of the device portal for your Oracle9*i*AS Wireless Server installation.
3. Login into the Oracle9*i*AS Wireless Portal with your studio username and password.

4. Select the MyHelloWorld application.
   The Oracle9*i*AS Wireless Server retrieves the helloworld.xml page and displays it.
5. You can debug the MyHelloWorld application by looking at the log file from you Studio web page. To view the log:
   - go to the Studio web site and login with you studio username and password
   - Select the Log icon for the MyHelloWorld application. The system pops up the log viewer on the web site. The log viewer displays the content retrieved.

9.1.1.4 Deploy the HelloWorld Application

To deploy an application you will must be part of a Domain. In this walkthrough you will create a Domain and then deploy the MyHelloWorld application to an Oracle9*i*AS Wireless Server.

1. From the Studio Menu select MyDomains.
2. Enter the following in the Create Domain Window

   Name: SampleDomain
   Enter a password and confirm the password selection.
   Set as Default should be checked.
3. Click on MyStudio on the Studio Menu.
4. Select the MyHelloWorld application and select deploy.
5. On the Deploy Application Page, click the Deploy button. This will deploy the application on to the Domain Host defined by your administrator.

9.1.2 Oracle Online Mobile Studio

You can also use OracleMobile's *hosted* Online Studio (a developer portal) to quickly and easily build, test and deploy your mobile applications.

OracleMobile's *hosted* Online Studio enables any developer, systems integrator or independent software vendor to quickly develop a mobile application that is immediately accessible from any device. This unique, next generation environment allows companies to benefit from faster time to market, increased productivity, and a dramatically simplified testing cycle, while providing access to the latest mobile applications and tools. It enables you to focus on your business logic which is your core competency, while we focus on the device complexity, our core competency.

For more information on Oracle Online Mobile Studio, and to use OracleMobileOnline Studio, see Oracle Technology Network (OTN) at: http://otn.oracle.com.

9.2 Oracle9*i*AS Wireless SDK

9.2.1 Overview

Oracle9*i*AS Wireless SDK is a light development version of Oracle9*i*AS Wireless. It is an off-line environment that enables developers to create and test MobileXML applications. With Oracle9*i*AS Wireless SDK, application developers can test and simulate applications without needing to support a complete Oracle9*i*AS Wireless installation.

The SDK can be separated into two sub-components: transcoder and messaging.

Oracle9*i*AS Wireless SDK transcoder can be used to test Mobile XML applications, new XSL stylesheet transformers and new device descriptions. It provides most of the functionality available in Oracle9*i*AS Wireless device portal.

Oracle9*i*AS Wireless SDK messaging API is the same as PushAPI, which delivers all kinds of messages through Push server or Push Messaging gateway. Push Messaging gateway is based on SOAP technology. Push server is build on top of Oracle9*i*AS Wireless. The implementation of Push Server has been simplified in Mobile SDK so that SDK does not have dependencies on SOAP or Oracle9*i*AS Wireless. Only a Push Server simulator is shipped with SDK, which can send out emails only. All messages with other transports will be converted to email and send to an email address called 'default email'. Therefore, a SMTP email server and a valid email address are needed.

9.2.2 Installation

Oracle9*i*AS Wireless SDK is a J2EE application. It should be deployed on Oracle9*i* Application Server using the OC4J deployment tool. For more details please refer to the OC4J documentation.

9.2.3 Structure

Oracle9*i*AS Wireless SDK contains the following important files and directories:

- **index.jsp** - this is the single entry point to the SDK from all devices. Depending on the device capabilities, i.e. PC- on non-PC-browser, it redirects the request to the appropriate URL.
- **index.html** - this is the SDK main page when accessed from a PC-browser. It contains useful links to the SDK test transcoder, the readme document (this file), the admin page, JavaDoc, and the Mobile XML documentation.
- **Home.jsp** - this is the SDK default main wireless application (configurable in WEB-INF/web.xml). See Default Main Wireless Application for details.
- **apps** - this directory contains the default entry points for all example applications (configurable in WEB-INF/web.xml). See Default Main Wireless Application for details.
- **docs** -Wireless SDK developer documentation.
  - **javadoc** - JavaDoc for the Oracle9*i*AS Wireless SDK messaging API.
  - **mxml** - mobile XML documentation.
- **examples** - example applications using the messaging API.
- **logs** - this is the default SDK log directory (configurable in WEB-INF/web.xml).
  - **omsdk.log** - the default SDK log file (configurable in WEB-INF/web.xml). The SDK uses a single log file, without overriding it. If you want to remove the old log file and start using a new one, you must delete the old file manually.
- **repository** - this directory (including all subdirectories) is the SDK repository (configurable in WEB-INF/web.xml. NOTE: If you want to move this directory somewhere else you have to preserve the subdirectories structure). Every .xml file in this directory is considered a separate device description. If you want to add support for a new device simply add a new XML-file with the description of the new device.
  - **XFORM** - this directory contains subdirectories with XSL device transformers. The name of the subdirectories must be in this format m.n where m is the major version number and n is the minor version number of the mobile XML schema that the XSL transformers support

    1.1 - XSL device transformers for version 1.1 of the mobile XML schema. Every .xsl file in this directory is considered a separate transformer. The name of the file is the name of the transformer (the names are case sensitive). There are two special files in this directory:

SimpleResult_1_1_0.xsd - this is the XML schema describing Mobile XML version 1.1

SimpleResult_1_1_0.dtd - this is the XML DTD describing Mobile XML version 1.1

- **WEB-INF**
  - **web.xml** - this is the main configuration file for the Oracle9*i*AS Wireless SDK. Some of the entries in this file can be modified to change the default behavior of the SDK.
  - **classes/messages** - this directory contains localized messages used by the XSL transformers.
  - **classes/oracle/panama/core/admin**

    EncodingSets.properties - mappings between IANA and Java character encoding names. Normally you should not have to modify this file.

    ProxyFirewall.properties - proxy firewall settings for the SDK. By default the SDK is not configured to use a firewall. You should modify this file only if you need to access an application that is outside your firewall.
- **lib**
  - **omsdk.jar** - Oracle9*i*AS Wireless SDK run-time engine implementation. This file includes also the messaging API. You will need to include it in the CLASSPATH if you build applications using the SDK messaging API.

9.2.4 Configuration

9.2.4.1 SDK Transcoder

Oracle9*i*AS Wireless SDK transcoder is a J2EE web application. The configuration file for the application is WEB-INF/web.xml. Some of the configuration properties can be modified at run-time using the SDK administration page.

Note: The changes made from the SDK administration page do not get persisted in web.xml. You will have to modify web.xml manually if you want to use the new settings permanently.

Here is the list of user configurable settings:

- **omsdk.repository.dir** - absolute path to the SDK repository directory (use either "/" or "\\" as file separator on Windows). By default the value for this context parameter is not set. The SDK will assume that the repository is in [SDK-context-root-directory]/repository.
- **omsdk.apps.dir** - absolute path to the default applications directory (use either "/" or "\\" as file separator on Windows). By default the value for this context parameter is not set. The SDK will assume that the applications directory is [SDK-context-root-directory]/apps.
- **omsdk.log.file** - absolute path to the SDK log file (use either "/" or "\\" as file separator on Windows). By default the value for this context parameter is not set. The SDK will create a new file omsdk.log in [SDK-context-root-directory]/logs directory. The log file can be viewed from the SDK administration page. You can redirect the log information to the system output or error (in this case you cannot see the log data from the administration page). The SDK does not delete the log every time it is written. It continues to use the same log file. If you want to start with an empty log file, delete the old one or change this value to use a different file name.
- **omsdk.log.level** - log level (can be modified from the administration page). There are four log levels: debug, info, warn and error. Levels are inclusive, that is, warn level displays any log message marked as warning or error. Default level is info.
- **xml.validation.mode** - used to validate the mobile XML received from the application (can be modified from the administration page).

  Sets the validation mode of the XML parser to one of these four types: [schema | dtd | partial | none]. Default value is schema.

  Note: In order to use XML parser validation, you must provide an XSD file (for schema) and a DTD file (for DTD) validation. These files must reside in the same directory as the corresponding XSL files. The SDK comes with SimpleResult_1_1_0.xsd and SimpleResult_1_1_0.dtd files.

- **autoreload.transformers** - this flag enables/disables autoreloading of the transformers. If autoreloading is enabled, the SDK checks the timestamp of the transformer XSL file and automatically reloads the file if it was modified. The values for this setting are true and false. Default value is true.

- **autoreload.devices** - this flag enables / disables autoreloading of the device descriptions. It has the same meaning for the devices as autoreload.transformers does for the transformers. Default value is true.
- **home.page.url** - URL to the main wireless application. This is the application that will be invoked when a device sends a request to the transcoder. The value could be either absolute or relative URL. If a relative URL is used it must be relative to the Oracle9*i*AS Wireless SDK context.

WARNING: Do not modify the remaining settings in WEB-INF / web.xml.

9.2.4.2 Properties Files

The SDK transcoder uses two properties files for additional configuration. Normally you should not need to modify those files.

- **EncodingSets.properties** - mappings between IANA and Java character encoding names. You will have to modify this file only if you have problems with some locale-specific characters. The file contains a brief explanation how to add new entries.
  - The IANA character set names are published here: http: / / www.iana.org / assignments / character-sets
  - This Java encoding names are published here: http: / / javasoft.com / j2se / 1.3 / docs / guide / intl / encoding.doc.html
- **ProxyFirewall.properties** - proxy firewall configuration parameters. You will need to modify this file only if you need to access Mobile XML applications that are outside your firewall. The file contains explanation about every entry.

9.2.5 SDK Messaging

9.2.5.1 Prerequisites

- JDK 1.2 or above
- SMTP email server
- Valid email address as default email address
- Java mail jars from JavaSoft: mail.jar, activation.jar

9.2.5.2 Configuration Parameters

Required Parameters

The SDK reads SMTP mail server and default email from the Java VM System properties. Property name for SMTP server is mobile-sdk.email.server.host and property name for default email is mobile-sdk.default_email. These two parameters are required to run SDK.

There are two ways to set those parameter:

1. Programatically
   - In your application that you need to call
   - System.setProperty("mobile-sdk.email.server.host","smtp.company.com");
   - System.setProperty("mobile-sdk.default_email","default-email@company.com"); before the first call to the messaging API
2. Passing command line parameters to the Java VM
   - java -Dmobile-sdk.email.server.host=smtp.company.com -Dmobile-sdk.default_email=default-email@company.com.

Not Required Parameters

MessagingGatewayURL, username and password of the constructors of Push and PushLite classes are not going to be used. But, you have to pass something to construct the instance. Passing three nulls will be permissible.

9.2.5.3 Push and PushLite

Push and PushLite have the same functionality: deliver messages to Push server (Push Server simulator in SDK). But, why do we need them both?

Push takes an instance of Packet as parameter. Packet has a message object, senders, recipients and additional information, which helps to deliver the message. For example: priority, speed of delivery, delay etc.

PushLite can send out text messages only. It's very easy to use. The users don't need to know any other classes like Packet, Message, and AddressData etc.

The reason that they co-exist in the same API is because Push and PushLite give developers an opportunity to choose the API to meet their needs. If you want to send text messages fast, use PushLite. If you need more control over the message, use Push.

The source files for both classes are in the examples directory.

9.2.6 Device Description

Oracle9*i*AS Wireless SDK stores the device description as XML files in its repository directory. Each XML file stores the description of one device. The XML root element is <LDEV>. All element attributes and subelements are exactly the same as in the Oracle9*i*AS Wireless repository XML representation (there are a few exceptions that will be explained below). The advantage of that is that you can download the Oracle9*i*AS Wireless repository, save all <LDEV> elements from the repository XML file as separate files, copy them into the SDK repository directory (or modify web.xml file), and the SDK will use exactly the same device descriptions as your real server. Or, you can add a new device description to the SDK repository, fully test it, and then deploy it on your real Oracle9*i*AS Wireless server. In order to upload a new device to the real Oracle9*i*AS Wireless server you will need to create a new XML file with the following format:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<PanamaObjects>
  <LDEV_LIST>
    <LDEV ...>
          The new device description goes here
        </LDEV>
  </LDEV_LIST>
</PanamaObjects>
```

Where the <LDEV> element is the one stored in the XML file in the SDK repository directory.

Here is the complete list of attributes and subelements that comprises the LDEV element. Please keep in mind that all String values are case sensitive.

9.2.6.1 Attributes

name - a String - the name of the device. The value of this attribute must be the same as the name of the XML file (without the .xml extension) in which the device description is stored. The value must be unique.

mimeType - a String - the MIME type that the device expects, for example: "text/vnd.wap.wml" or "text/html"

encoding - a String - the content encoding. The IANA character set names are published at:

http://www.iana.org/assignments/character-sets

The "mimeType" and the "encoding" attributes are used to create the Content-Type HTTP header that is sent back to the end user device. For more details please see the HTTP 1.1 specification at:

http://www.rfc.net/rfc2616.html

9.2.6.2 Example

```
Content-Type: text/vnd.wap.wml; charset=ISO-8859-1
```

deviceCategory - a String - the device category. Oracle9*i*AS Wireless groups all devices in the following six categories:

- pcbrowser
- pdabrowser
- microbrowser
- messenger
- micromessenger
- voice

See Chapter 10, "Core Technologies" for more details of the different device classes.

manufacturer - a String - the company name. For example: Nokia, Ericsson, Palm, Motorola, etc.

model - a String - the device model.

softKeys - an Integer - the number of soft keys that the device has.

screenCols - an Integer - the number of characters (per row) that the device can display.

screenRows - an Integer - the number of rows text data that the device can display.

screenWidth - an Integer - the screen width in pixels.

screenHeight - an Integer - the screen height in pixels.

imageCapable - a Boolean - whether the device supports images or not.

colorCapable - a Boolean - whether the device supports colors or not.

bitsPerPixel - an Integer - the number of bits per pixel used to represent either the color or the gray scale.

videoCapable - a Boolean - whether the device supports streaming video or not.

voiceCapable - a Boolean - whether the device supports voice or not.

system - a Boolean - whether this is a "system" device or not.

maxDocSize - size of document (in bytes) that a device can accept.

supportsAmpersandEntity - ampersand character can be used in XML-friendly devices.

supportsRelativeURL - a Boolean - whether the device supports relative URLs or not. In general all browsers should resolve relative URLs but of them do not do it.

prolog - xml prolog at the start of the content sent to a device. Specifies content type.

description - a String - a short description of the device.

needsURLCaching - a Boolean - whether the URLs for this device should be cached or not.

> **Note:** This attribute has been deprecated. Both, the Oracle9*i*AS Wireless SDK and the Oracle9*i*AS Wireless server will always cache the URLs for all devices.

supportsCookie - a Boolean - whether the device supports "cookies" or not.

defaultTransformer - a String - the name of the XSL transformer to be used for this device.

> **Note:** This attribute has been deprecated in the Oracle9*i*AS Wireless server. It has been replaced by the "Transformers" subelement which contains the list of transformers to be used for the different versions of the Mobile XML. In the current version the SDK supports only a single version of the Mobile XML language. And it reads the default Transformer attribute for the transformer to be used.

9.2.6.3 Subelements:

- **UserAgents** - the list of HTTP User-Agent headers that should be mapped to this device.
  - **UserAgent** - its "value" attribute stores individual User-Agents. You can use "*" wildcard to map zero or more characters.

- **ImageFormatPreferences** - the list of image formats that the device supports.
  - **ImageFormatPreference** - its "mimeType" attribute stores individual MIME type value and the file extension. For more information about Internet media types, please read RFC 2045, 2046, 2047, 2048, and 2077. The Internet media type registry is at ftp://ftp.iana.org/in-notes/iana/assignments/media-types/
- **VideoFormatPreferences**
  - **VideoFormatPreference**
- **Transformers** - the list of transformers to be used with this device

  Note: This subelement is not used by the SDK.

  - **Transformer** - its "name" attribute stores the name of a single transformer.
- **EXT_ATTR** - deprecated subelement that used to store the device login and error pages. Not used by the SDK.

9.2.7 Deploy the HelloWorld Application

To deploy an application you will must be part of a Domain. In this walkthrough you will create a Domain and then deploy the MyHelloWorld application to an Oracle9*i*AS Wireless Server.

1. From the Studio Menu select MyDomains.
2. Enter the following in the Create Domain Window

   Name: SampleDomain
   Enter a password and confirm the password selection.
   Set as Default should be checked.
3. Click on MyStudio on the Studio Menu.
4. Select the MyHelloWorld application and select deploy.
5. On the Deploy Application Page, click the Deploy button. This will deploy the application on to the Domain Host defined by your administrator.

9.2.8 Device Detection

Oracle9*i*AS Wireless SDK uses the same device detection mechanism as Oracle9*i*AS Wireless server. See Chapter 10, "Core Technologies" for more details.

9.2.9 Default Main Wireless Application

Oracle9*i*AS Wireless SDK comes with a default demo wireless application. This application is a single JSP page: Home.jsp. This JSP page looks in the apps directory to find user-specific applications. A user application can be a .jsp, .xml or .mxml file. The main wireless application displays a link to every one of the files it finds in the apps directory. If you want to test your application, copy its main page into the apps directory. If your application contains more than one file, then only the first page must be copied into the apps directory. All other pages should be in a separate directory (it could be a subdirectory of the apps directory). See the sample applications for more details.

9.3 Overview of JDeveloper with Oracle9*i*AS Wireless

JDeveloper provides a mechanism to develop, debug and test Oracle9*i*AS Wireless XML JSPs and XML pages in a single tool by providing an Oracle9*i*AS Wireless addin for JDeveloper. Developers can create JSP pages with embedded BC4J data tags with Oracle9*i*AS Wireless XML tags and by invoking these servlets through OC4J, they can run any BC4J application on a wireless device emulator. By using the power of the schema-driven editor, developers can create Oracle9*i*AS Wireless XML pages that they can further call from their JSP pages. Based on the source of the device request, the correct device stylesheet is applied to the XML document. Figure 9–1, "Simplified Request Path" shows how the servlet works (and could use BC4J as a data source for example).

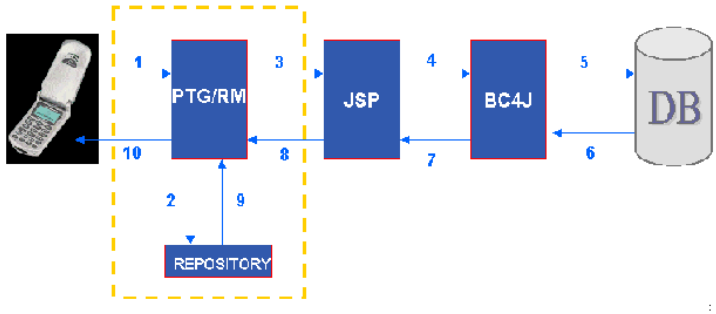
To maximize developer productivity, JDeveloper provides a comprehensive set of integrated tools to support the complete development lifecycle, from source control, modeling, and coding through debugging, testing, profiling, and deploying. JDeveloper simplifies J2EE development by providing wizards, editors, visual design tools, and deployment tools to create high-quality, standard J2EE components including applets, JavaBeans, JavaServer Pages (JSP), servlets, and Enterprise JavaBeans (EJB). JDeveloper also provides a public Addin API to extend and customize the development environment and to seamlessly integrate with external products.

To simplify the development of scalable, high-performance J2EE applications, JDeveloper offers an open and extensible J2EE framework called Business Components for Java (BC4J). BC4J is an object-relational mapping tool that implements Sun's J2EE design patterns, allowing developers to quickly build sophisticated J2EE applications.

9.3.1 JDeveloper and Oracle9*i*AS Wireless SDK

The SDK primarily consists of a Java servlet which links to a repository of stylesheets. Based on the source of the HTTP request, the correct stylesheet is applied by the servlet to transform the data to the target markup language for that device. By providing an addin for integrating the SDK into JDeveloper, we provide the developer a mechanism to develop, debug and test Oracle9*i*AS Wireless XML JSPs and XML pages in a single tool. The developer can create JSP pages with embedded BC4J data tags and Oracle9*i*AS Wireless XML tags and by invoking these servlets through OC4J, they can run any BC4J application on a wireless device emulator. By using the power of the schema-driven editor they can create Oracle9*i*AS Wireless XML pages that they can further call from their JSP pages. Figure 9–1, "Simplified Request Path" demonstrates how the servlet works (and could use BC4J as a data source for example):

Figure 9–1 Simplified Request Path



1. A request is received from a wireless client (which is routed from the gateway to the application server) for an Oracle9*i*AS Wireless JSP page. The structure of such a JSP is as follows:

```
<%@ page language="java" import="oracle.jbo.*" contentType="text/vnd.oracle.iAS
Wireless XML;charset=WINDOWS-1252" %>
<%@ taglib uri="/webapp/DataTags.tld" prefix="jbo" %>
<SimpleResult>
<SimpleContainer>
<SimpleText>
<SimpleTextItem>Browse Form</SimpleTextItem>
</SimpleText>
<jbo:ApplicationModule id="am"
configname="mypackage.MypackageModule.MypackageModuleLocal"
releasemode="Stateful" />
```

```
<jbo:DataSource id="ds" appid="am" viewobject="DeptView" rangesize="3"/>
<jbo:DataHandler appid="am" />
<SimpleText>
<SimpleTextItem>DeptView Browse Form</SimpleTextItem>
</SimpleText>
<SimpleTable>
<SimpleTableBody>
<SimpleRow>
<SimpleCol><jbo:DataScroller datasource="ds" /></SimpleCol>
<SimpleRow>
<SimpleRow>
<SimpleCol><jbo:DataTable datasource="ds" /></SimpleCol>
</SimpleRow>
</SimpleTableBody>
</SimpleTable>
<SimpleContainer>
<jbo:ReleasePageResources />
</SimpleResult>
```

2. The application server then launches the REQUEST MANAGER (RM) servlet which then handles the requests coming from the client. As a part of its init() process, the RM servlet looks for the repository containing the XSLT stylesheets and loads it up into memory.
3. The servlet then executes the JSP page (the JSP page may have BC4J data tags) that are then interpreted and executed. The data is populated in the page with SimpleResult tags.
4. The RM servlet now received this XML page.
5. It then applies the correct transformation to the data received based upon the content type and the source of the HTTP request.
6. Finally the correct markup is sent back to the client where the request originated.

9.3.2 The Addin and the Wizards

There are two simple wizards which allow a user to create an Oracle9*i*AS Wireless JSP and an Oracle9*i*AS Wireless XML document respectively.

The execution flow is as follows:

1. Create a new Oracle9*i*AS Wireless JSP by going through the following steps:

a. Choose File | New | Web Objects | 9iAS Wireless JSP Wizard. The Oracle9*i*AS JSP Wizard is launched.

b. Specify a name for the JSP (or choose the default)

c. Select if you want to generate code for a form or menu (or both) and click OK. A new Oracle9*i*AS WE JSP is created. In addition this automatically performs the following actions:

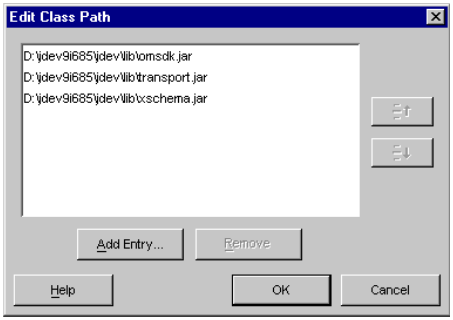
– Updates the `web.xml` file with the relevant servlet information (servlet name, class, parameters, etc.) and adds it to the current project. Here is the web.xml file:

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.2//EN" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
<display-name>Oracle 9iAS App</display-name><description>Oracle 9i Application
Server Wireless SDK Application</description>
<context-param>
<param-name>omsdk.repository.path</param-name>
<param-value>D:\OMSDK\repository</param-value>
</context-param>
<context-param>
<param-name>omsdk.apps.path</param-name>
<param-value>D:\OMSDK\apps</param-value>
</context-param>
<context-param>
<param-name>omsdk.log.path</param-name>
<param-value>D:\OMSDK\logs</param-value>
</context-param>
<servlet>
<servlet-name>sdk</servlet-name>
<servlet-class>oracle.panama.sdk.SdkServlet</servlet-class>
<init-param>
<param-name>xml.validation.mode</param-name>
<param-value>none</param-value>
</init-param>
<init-param>
<param-name>log.level</param-name>
<param-value>debug</param-value>
</init-param>
<init-param>
<param-name>autoreload.transformers</param-name>
<param-value>true</param-value>
</init-param>
```

```
<init-param>
<param-name>autoreload.devices</param-name>
<param-value>true</param-value>
</init-param>
<init-param>
<param-name>home.page.url</param-name>
<param-value>Home.jsp</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet>
<servlet-name>lfv</servlet-name>
<servlet-class>oracle.panama.sdk.util.LogFileViewer</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>sdk</servlet-name>
<url-pattern>/rm</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>lfv</servlet-name>
<url-pattern>/log</url-pattern>
</servlet-mapping>
<session-config>
<session-timeout>30</session-timeout>
</session-config>
<mime-mapping>
<extension>mxml</extension>
<mime-type>text/vnd.oracle.iAS Wireless XML</mime-type>
</mime-mapping>
<mime-mapping>
<extension>log</extension>
<mime-type>text/plain</mime-type>
</mime-mapping>
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
<welcome-file>index.html</welcome-file>  </welcome-file-list>
</web-app>
```

– Adds the relevant libraries to the classpath as shown below:

Figure 9–2 Classpath



– Creates the following JSP file as shown below (with the relevant content-type):

```
<?xml version="1.0" encoding="WINDOWS-1252" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1.0//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<%@ page contentType="text/vnd.oracle.iAS Wireless XML; charset=WINDOWS-1252"
%>
<%@ page language="java" %>
<%@ page import='java.util.*' %>
<Simpleresult>
<Simplecontainer>
<Simpletext>
<Simpletextitem>
The current time is <% out.println((new java.util.Date()).toString()); %>
</Simpletextitem>
</Simpletext>
<menu>
<choice next="choice1.jsp">Choice 1</choice>
<choice next="choice2.jsp">Choice 2</choice>
<choice next="choice3.jsp">Choice 3</choice>
</menu>
</Simplecontainer>
</Simpleresult>
```

Note: The content-type of this page is text/vnd.oracle.*i*AS Wireless XML, NOT html.

2. Since this is a JSP page, you can include BC4J data tags in the page that means you can data enable it. However, instead of having HTML tags like a typical JSP would have, this page would have Oracle9*i*AS Wireless XML tags (which makes it akin to a data-bound UIX page). Assuming that the page is syntactically correct, when you run this page by right-clicking on it, the following steps occur:
   - It launches the embedded OC4J,
   - Invoke the correct servlet(s) and
   - Launches the user's default browser and pass the URL based on the application's root context.

The difference between a typical JSP and the Oracle9*i*AS Wireless JSP is that the former is automatically *run-able* by any servlet engine, where as the latter is more like an XML document which needs to be processed by a servlet. In this case, its behavior is similar to a UIX page.

9.3.3 Instructions to use the Addin and Wizards

9.3.3.1 Installation Steps

Here are the instructions for installing Oracle9*i*AS Wireless Wizards and configuring the JDeveloper properties file to run the addin and the wizards.

1. Download the addin zip file from http://otn.oracle.com (in the “products”, then “Jdeveloper” section) and unzip it to a directory (for example: **D:\omsdkAddin**).
2. Add the project's output path to the $[JDEV_HOME]\bin\jdev.conf file (for example: AddJavaLibFile ../../../../classes)

   Note: This (AddJavaLibFile) needs to be at the very end of the file, not anywhere else!

3. Add this project's main Addin to the $[JDEV_HOME]\bin\addins.properties file. For example:
   - AddinCount=100
   - Addin99=oracle.iaswe.iasWEAddin
4. Add the *i*AS Gallery elements to $(JD9i)\lib\gallery.xml file.

5. Find the element called Web Objects and update it to read as follows:

```
<Item class="oracle.ide.gallery.GalleryElement">
<name>9iAS WE XML Wizard</name>
<wizardClass>oracle.iaswe.iasWEXMLWizard</wizardClass>
<wizardParameters/>
</Item>
<Item class="oracle.ide.gallery.GalleryElement">
<name>9iAS WE JSP Wizard</name>
<wizardClass>oracle.iaswe.iasWEJSPWizard</wizardClass>
<wizardParameters/>
</Item>
```

9.3.4 Running Instructions

1. Run JDeveloper
2. Create a New Project for Testing
3. Select Menu File | New ...
4. Click on the Web Objects | *i*AS WE JSP Wizard and create the JSP file
   - Notice Web.xml was added to project
   - Double click on the Project to see the libraries added to project
   - Look at the contents for correctness
5. Click on the Oracle9*i*AS XML Wizard and create the xml file similarly.
   - Look at the contents for correctness
   - Right click on these nodes to notice the option
   - Click on the Run menu to see the options
6. Now you can add the BC4J data tags in your JSP pages to access the data in the application logic tier.

9.4 Third-party Mobile Simulators

Although you will be able to test you mobile applications using a regular Web Browser on your personal computer, it is recommended that you perform testing using various device emulators with different form factors. This will allow you to understand the constructs on Oracle9*i*AS Wireless XML with respect to rendition on varying device form factors.

Various mobile browser vendors have emulators available that can run on a typical desktop environment. This section lists mobile browser emulators available, categorized into different form factors. The list below is a sample, and provides an introduction to various mobile simulators available; it is not an exhaustive list of all emulators available.

9.4.1 Phones

The typical phone device is considered to have a small form factor, although there are phones in the market that support form factor and functionality similar to a PDA device. The browser simulators that support relatively small form factors include Nokia6210, and Phone.com's HDML and WML simulators. Below is a list of phone browser simulators that can be used to test you Oracle9*i*AS Wireless applications. These simulators run on your Personal computer, and connect to the Oracle9*i*AS Wireless server over HTTP protocol.

9.4.1.1 Openwave SDK 3.2

This is an HDML (HandHeld Markup Language) simulator provided by Openwave. You can simulate application behavior on phones that support HDML browsers. HDML is a proprietary markup language supported by Openwave browsers only.

> **Note:** This version of SDK can support both WML and HDML. Ensure that Oracle9*i*AS Wireless is generating HDML for requests from this simulator. You can use the web tool and configure Oracle9*i*AS Wireless to generate HDML for requests from this simulator. Openwave SDK is provided by Openwave Systems Inc. For more information go to http://developer.openwave.com.

9.4.1.2 Openwave SDK 4.1 and 5.0

You can use SDK 4.1 to simulate your application with WML 1.1 Openwave browsers and SDK 5.0 to simulate with WML 1.3 Openwave browsers. Openwave SDK is provided by Openwave Systems Inc.For more information go to http://developer.openwave.com.

9.4.1.3 Nokia Mobile Internet Toolkit

This toolkit is provided by Nokia and has a simulator for Nokia's WML browser. You can simulate your application on different Nokia phones. The Mobile Internet

Tool Kit is provided by Nokia Corporation. For more information go to http://www.forum.nokia.com.

9.4.1.4 Ericsson's WapIDE 3.1.1 SDK

This SDK is provided by Ericsson and allows you to simulate WML applications on various Ericsson phones. The WapIDE 3.1.1 SDK is provided by Telefonaktiebolaget LM Ericsson. For more information see www.ericsson.com.

9.4.1.5 Yospace Simulator

Yospace provides various WAP simulators that can used to test your application experience on various WML browsers. Yospace simulator is provided by Yospace Holdings Ltd. For more information go to http://www.yospace.com.

9.4.2 PDA

The typical PDA device is considered to have a medium form factor. The form factor of PDA is higher than that of a typical phone. The simulators that support PDA-style devices are PocketPC and PalmOS simulators. There are other simulators that support PDA style form factor and also other phone devices with a PDA form factor.

9.4.2.1 Palm OS Simulator

Simulates the Palm OS on your personal computer. You can typically install on to the Palm OS simulator a PQA or browsers such as Eudora. This enables you to test and simulate you application behavior on a PalmOS. Palm, Inc. provides this simulator. For more information see www.palmos.com.

9.4.2.2 PocketPC SDK

The PocketPC SDK is a desktop application and contains a PocketPC simulator that runs on your personal computer. You can use a browser application on PocketPC such as Pocket Internet Explorer, or similar, to test your application on a PocketPC device. Microsoft Corporation provides PocketPC SDK. For more information go to http://www.microsoft.com.

9.4.3 Voice

Voice device are classified as a separate form factor. This is because voice devices, unlike other data devices, do not allow the user to scan the entire document. On

voice devices, the user must wait until the voice browser reads the entire document; it is also difficult for users to “scroll” the document.

9.4.3.1 IBM Voice Server SDK

IBM provides a Voice Server SDK running on a personal computer, and supports VoiceXML technology. You can use IBM’s Voice Server SDK to test your VoiceXML applications. The Voice Server SDK is provided by IBM Corporation. For more information see www.ibm.com.

9.4.3.2 VoiceGenie

VoiceGenie hosts a developer Voice Gateway that allows you to test your applications over voice. Also VoiceGenie provides *Genie IDE* that simulates the Voice platform to test your applications. VoiceGenie Technologies Inc. provides both the developer voice gateway and the Genie IDE. For more information see developer.voicegenie.com.

9.5 Deploying Your Applications

Oracle9*i*AS Wireless provides Web-based, role-specific tools to create, manage, and deploy mobile services. These webtools include wizards for developing and managing repository objects, and utilities for managing the server and deploying Oracle9*i*AS Wireless.

After creating your applications, use Oracle9*i*AS Wireless webtools to deploy them to your customers using your Wireless instance.

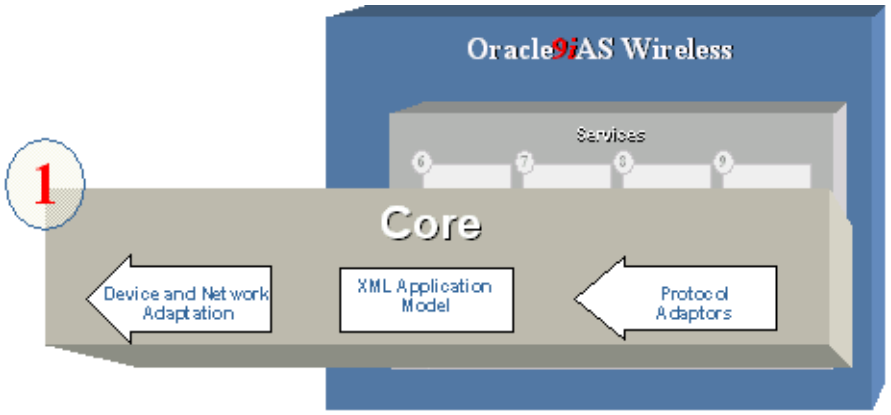
For more information on these web-based tools, see *Oracle9i Wireless Getting Started and System Guid*e.

10

Core Technologies

This chapter discusses how you can use the Oracle9*i*AS Wireless 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 Oracle9*i*AS Wireless, and work with the XML formats that the Oracle9*i*AS Wireless uses. Sections include:

Figure 10–1 Core technologies

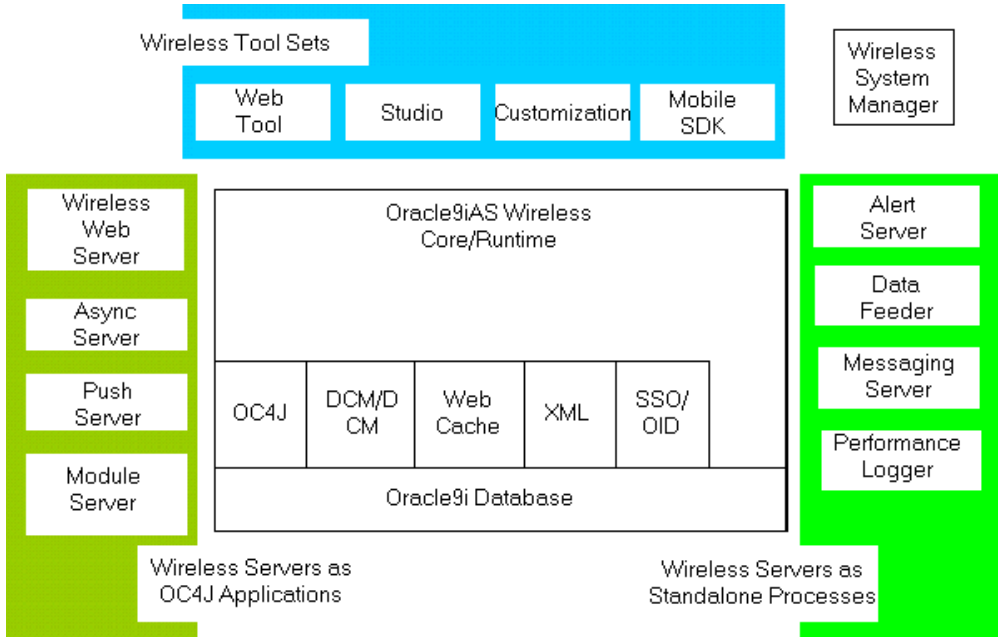


10.1 Oracle9*i*AS Wireless Components and Process Architecture

10.1.1 Core Platform Architecture

Oracle9*i*AS Wireless provides a powerful, complete and integrated platform for developing, testing and deploying mobile applications. The Oracle9*i*AS Wireless core, runtime, tools are built top of proven Oracle technologies including OC4J Container, Distributed Configuration Management (DCM), Enterprise Management Daemon (EMD), XML, Oracle Internet Directory (OID), Single Sign-On Server (), Oracle Process Manager (OPMN), WebCache, and Oracle9*i*. Oracle9*i*AS Wireless in-house and community development and testing tools make the mobile application development easier. Oracle9*i*AS Wireless Server can take mobile applications to be deployed to any mobile network, and accessible from any device through any gateway.

Figure 10–2 Oracle9iAS Wireless Platform Architecture



As depicted in the above diagram, Oracle9*i*AS Wireless provides the following wireless development / deployment tool sets:

- WebTool – provides an advanced in-house device and transformers management, mobile application development, testing, management and deployment, and mobile user management.

  See *Oracle9iAS Wireless System Guide and Getting Started* for more detail on how to use the webtools.

- Studio – provides simple mobile application testing and deployment in a developer community fashion.

  See Chapter 9, "Mobile Service Developer's Tools" for more detail on how to use it.

- Customization – provides an out-of-the-box testing and demonstration for mobile application customization through WEB. See *Oracle9iAS Wireless Getting Started and System Guide*, and Chapter 11, "Advanced Customization" for more detail on customization.

- Mobile SDK – provides a simple testing and debugging environment for mobile applications for developers without installing the entire Oracle9*i*AS Wireless software. The MobileSDK does not depend on Oracle9*i* database. JDeveloper add-ins can be downloaded from Oracle Technology Network so that MobileSDK can be integrated into the jDeveloper. Developing, debugging and testing a mobile application have been made easier.

  See Chapter 9, "Mobile Service Developer's Tools" for more detail on how to use it.

- Wireless System Manager – provides configuration management and performance monitoring for various wireless servers. It is packaged with Oracle9*i*AS Enterprise Manager, and launched through the Oracle9*i*AS Enterprise Manager console.

  See *Oracle9iAS Wireless Getting Started and System Guide* for more detail on how to use it.

- Wireless Servers deployed as OC4J applications:
  - Wireless Web Server – serves wireless requests through HTTP.
  - Async Server – servers wireless requests through non-HTTP, i.e. through email, SMS, and etc.

    See Section 10.5, "Asynchronous Server" in this chapter for more details.
  - Push Server – provides the capability to push a message to any device through any protocol.

    See Section 13, "Push Service and SMS" for more details.
  - Module Server – provides the built-in mobile applications in the areas of Personal Information Management (PIM), Mobile Commerce, and etc.

    See Chapter 18, "Mobile PIM and eMail" and Chapter 19, "m-Commerce" for more details.
- Other Wireless Servers deployed as standalone Java applications:
  - Alert Engine – provides alert services to subscribers.

    See Section 12.1, "Alert Engine" for more details.
  - Data Feeder – enables you to fetch content from content providers through any protocol in any format. The fetched content can be used as data source

for the alert engine or mobile applications.

See Section 12.2, "Data Feeders" for more details.

- Messaging Server – enable to deliver message in any protocol.

  See Section 10.1.2.1, "Key Execution Flows" for more details.

- Performance Logger – writes usage logging data of Wireless Web Server, Async Server, Messaging, Alert Engine and Data Feeder asynchronously to the database for performance monitoring purposes. Furthermore, the information stored in these tables can be utilized for business intelligence analysis.

10.1.2 Core Process Architecture

The following figure (divided into halves for easier viewing) shows how the above Wireless Platform and Tools are deployed physically in terms of processes and relationships with key components of a complete mobile application solution. The wireless-specific components are within the dark-blue rectangle. As the wireless component is an integral part of Oracle9*i*AS, it seamlessly integrates with other Oracle9*i*AS components including WebCache, Oracle Http Server, SSO, OID, EM, and Oracle Portal (highlighted with light-blue background color).

Figure 10–3 Oracle9iAS Wireless Process Architecture (part 1)

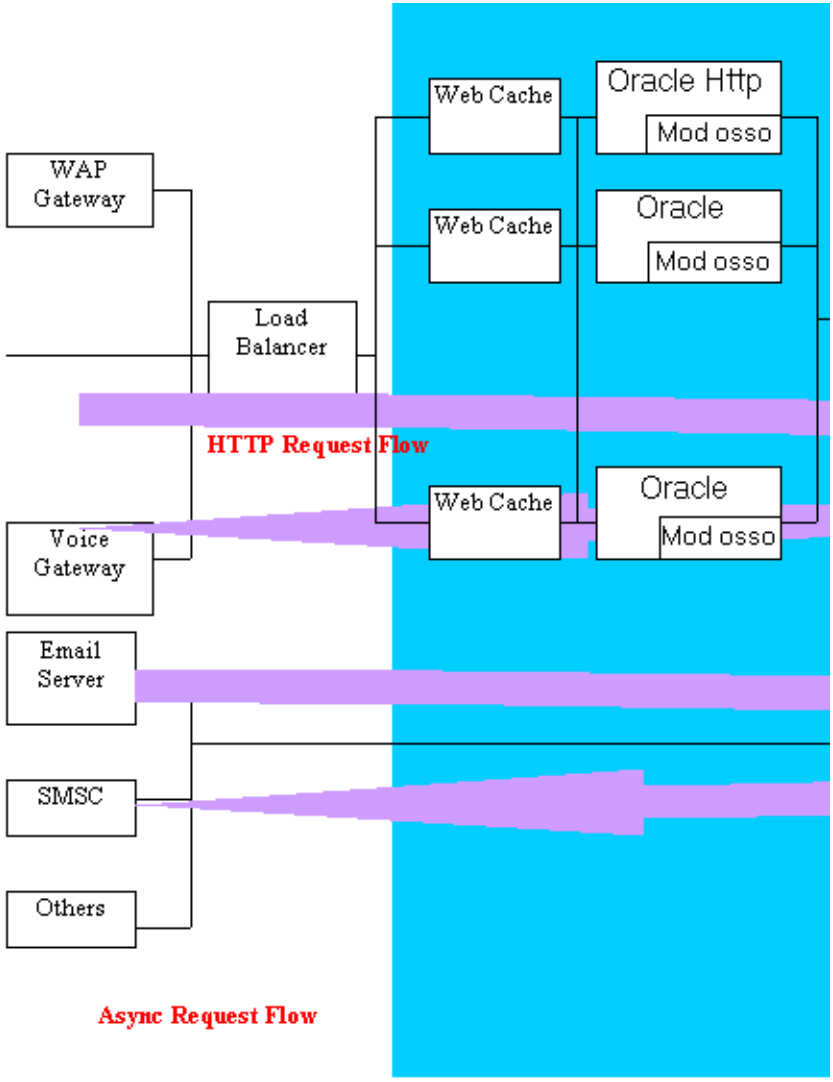
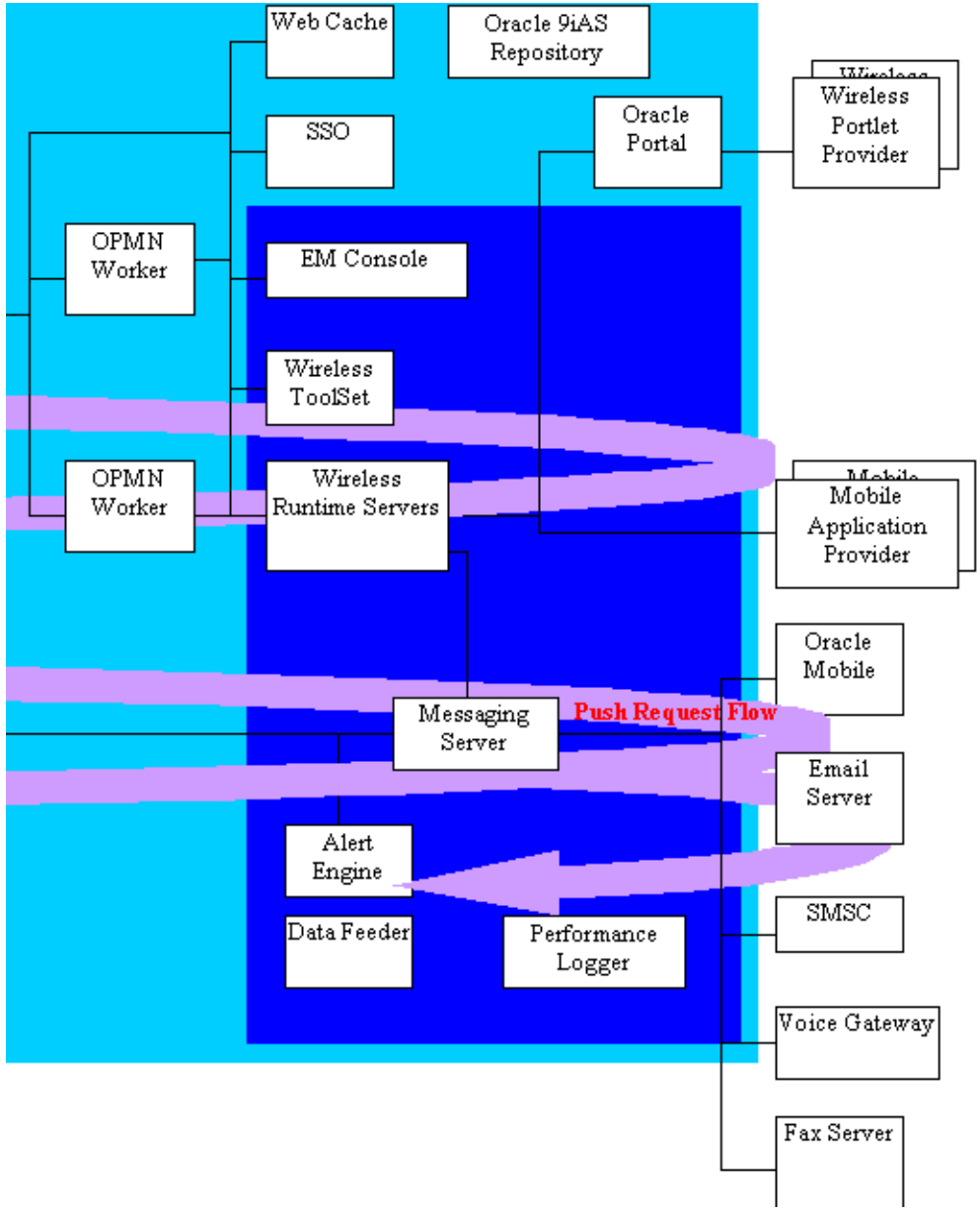


Figure 10–4 Oracle9iAS Wireless Process Architecture (part 2)



Oracle9*i*AS Wireless components contribute maximally 7 process groups on any machine on which the wireless component is installed and configured.

- Oracle Enterprise Manager (EM) Console – server that provides configuration management and performance monitoring for all Oracle9*i*AS components including Oracle9*i*AS Wireless. The wireless system manager is deployed on this server. There is one and only one EM server process allowed on any installed machine.
- Wireless ToolSet – All the wireless tools including WebTool, Studio, and Customization are deployed in this process group. This process group can be started or stopped through the OC4J manager in the EM console. Default installation will assign a single process to this process group. To increase scalability, additional processes can be assigned to this group by modifying the opmn.xml (REVISIT for the location and example). In this case, processes assigned to this group have the same configuration settings. By default, these OC4J applications are deployed, but are not auto-started until the first received request.
- Wireless Runtime Servers – All OC4J application-based wireless runtime servers including wireless web server, async server, push server, and module servers are deployed in this process group. This process group can also be started or stopped through OC4J manager in the EM console. The default installation will assign a single process to this process group. To increase scalability, additional processes can be assigned to this process by modifying the opmn.xml file (REVISIT for the location and example). In this case, processes assigned to this group have the same configuration settings. By default, these OC4J applications are deployed; only the wireless web server and async server are configured to be auto started; other applications are started upon the first received request.
- Wireless Standalone Java Processes – Alert Engine, DataFeeder, Performance Logger and Messaging Server are standalone Java Processes that can be started, stopped and configured through the wireless system manager (accessible through the EM console). The default installation only enables the performance logger. Ensure that Performance Logger has been started so that the performance of various wireless servers on this machine can be monitored through the wireless system manager. Other processes should only be started manually if their respective functionality is desired.

10.1.2.1 Key Execution Flows

Oracle9*i*AS Wireless platform can receive requests from any device via any protocol and deliver content to any device via any protocol. The key request execution flows are:

- Http Request Flow
- Async Request Execution Flow
- Push Request Execution Flow

Http Request Flow—Many devices with certain gateway support can request service through HTTP protocol. These devices include WAP phones with WAP gateways, fixed voice lines with VoiceXML gateways, and others. As illustrated in the above process architecture diagram:

1. Load Balancer dispatches a request sent from the external gateways to Oracle Http Server. Generally, Load Balancer supports sticky session; this means that the loader balancer will only load-balance these requests from a new session, otherwise the requests of an existing session will be delivered to the same Oracle HTTP Server. Load Balancer provides the hardware load-balancing solution.
2. Oracle HTTP Server dispatches the received request to OPMN Worker, or to the Wireless Web Server directly (based on the configuration). Requests are routed to OPMN worker (if OC4J-based software load balancing is desired and configured). Otherwise, the request is dispatched to the wireless web server directly.
3. OPMN worker dispatches the request to the appropriate process based on the process load (if the request is the first one of the current session). Otherwise, the OPMN worker dispatches the request to the wireless web server process to which the request session has been assigned.
4. The wireless web server processes receive the request. If the response for the request from this particular requesting device is cached by the WebCache, the response is returned immediately. If the request is to access a privileged service, then the wireless web server redirects the request to SSO. Otherwise it proceeds to step *b* below.
    a. SSO perform the sign-on process via the wireless web server process. After the sign-on succeeds, the original request resumes.
    b. Wireless web server dispatches the original request to the mobile application provider to request the mobile content in mobile XML.

5. The mobile application provide (which are the external mobile applications) process the request and return the mobile XML to the wireless web server process. Oracle Portal is just another mobile application provider.
6. Wireless web server adapts the received content to the network and device and returns to the request device.
7. The mobile content is visible on the requesting mobile device in its most native form.

Async Request Execution Flow—Wireless server can also process requests from non-HTTP based devices, such as SMS device, Pager, Email and etc. Here is the request execution flow:

1. Messaging Server receives a service invocation request message and dispatches it to the Async Server that runs insides the Wireless Runtime Server process.
2. Async Server preprocesses the request. The response is returned immediately. If the request is to access a privileged service, the wireless web server will redirect the request to SSO. Otherwise it proceeds to step *b* below.
    a. SSO performs the sign-on process via the wireless web server process. After the sign-on succeeds, the original request resumes.
    b. Wireless web server dispatches the original request to the mobile application provider to request the mobile content in mobile XML.
3. Async Server adapts the received response to the requesting device native format and sends the adapted response to Messaging Server.
4. Messaging Server dispatches the response to the requesting device.

Push Request Execution Flow—Wireless platform can also push any message to any device via different protocols. Out-of-the-box, any message can be pushed out as a SMS message, an email, a voice mail, a fax or to Oracle Mobile Message Gateway. The push request execution flow is as follows:

1. Push applications including Push Server, Alert Engine, or external applications can compose a message and send the message through calling push APIs.
2. Messaging Server asynchronously delivers the received message to the delivery provider through the specified protocol.
3. Messaging Server also asynchronously queries the delivery status (if supported by the provider).
4. Push applications can either pull the delivery status or be notified.

10.1.2.2 Default Configuration

The default installation configures the installed wireless component to work with the Oracle HTTP Server, WebCache on the local machine. The following mount points are added in the configuration file of the Oracle HTTP Server on the local machine:

/ptg -- for wireless web server

/async -- for async server

/modules – for module server

/webtool – for accessing webtools

/studio – for mobile studio

/customization – for accessing customization portal

/push -- for publishing the push message

If using an Oracle HTTP Server on a different machine (instead of on the local machine), you must manually configure the Oracle HTTP Server. For instructions on configuration, see *Oracle9iAS Wireless Getting Started and System Guide.*

By default, all the above mounting points are exposed. Comment out these mounting points (so you are not publishing the configuration file from the Oracle HTTP Server).

By default, the Wireless ToolSet and Wireless Runtime Server process groups are configured with single process only. See *Oracle9iAS Wireless Getting Started and System Guide* to learn how to configure them in load balancing mode.

10.1.2.3 Dependency

Files under ORACLE_HOME/wireless/lib belong to Oracle9*i*AS Wireless. They are:

- panama_modules.zip
- panama_modules_commerce.zip
- panama_modules_common.zip
- panama_modules_infra.zip
- panama_modules_location.zip
- panama_modules_pim.zip
- studio.jar

- wireless.jarclient.zip
- server.zip
- ssosdk902.jar

Oracle9*i*AS Wireless depends upon the following jar / zip files included in the Oracle9*i*AS Wireless common technology stack:

Table 10–1 Oracle9iAS Wireless Dependent Files

Depending jar/zip files	Description	Location
uix2.jar, share.jar	Uix	ORACLE_HOME / jlib / uix2.jar
ORACLE_ HOME / share / share.jar	classes12.zip	JDBC driver
ORACLE_HOME / jdbc / lib	jndi.jar	
ORACLE_HOME / jlib	xmlparserv2.jar	Xml parser
ORACLE_HOME / lib	sax2.jar, regexp.jar	
ORACLE_HOME / jlib	jai_codec.jar, jai_core.jar, jpeg_ codec.jar, ordimimg.jar	sdoapi.jar, sdovis.jar
Advanced imaging	ORACLE_HOME / ord / jlib	OH / lbs / mapviewer / web / WEB-INF / lib / sdoapi.jar
OH / lbs / mapviewer / web / WEB-INF / lib / sdovis.jar	providerutil.jar	LDAP provider
ORACLE_HOME / jlib	mail.jar, activiation.jar, pop3.jar	EMail client
ORACLE_HOME / lib	xschema.jar	
ORACLE_HOME / lib	http_client.jar, javax-ssl-1_2.jar, jssl-1_ 2.jar	http / ssl / https
OH / j2ee / home / lib / javax-s sl-1_2.jar	OH / j2ee / home / lib / jssl-1_2.jar	OH / lib / http_client.jar
dcm.jar, emd.jar, emPID.jar, log4j-core.jar	ORACLE_HOME / lib / libnmuk.so	ORACLE_HOME / bin / nmuk.dll
EM	OH / dcm / lib / dcm.jar	OH / sysman / webapps / emd / WEB-INF / l ib / emd.jar
OH / sysman / webapps / em d / WEB-INF / lib / log4-core.j ar	$ORACLE_HOME / lib / emPid.jar	ldapjclnt9.jar
OID client	OH / jlib / ldapjclnt9.jar	soap.jar

Table 10–1 Oracle9iAS Wireless Dependent Files

Depending jar/zip files	Description	Location
Soap	OH / soap / lib / soap.jar	repository.jar
Repository api	OH / jlib / repository.jar	ohw.jar
Oracle Help for Web		

10.2 Integration with other Components

This section describes Oracle9*i*AS Wireless integration with Single Sign-On (SSO) and Oracle Internet Directory (OID) server. The IAS v902 SSO is used by all IAS v902 components for user authentication, and OID is the single place for storing all the User related information.

This integration provides:

- a framework for secure SSO from browser clients to web-based applications, including Oracle Applications and Tools, through standard protocols.
- support for partner applications, which take full advantage of the SSO framework, as well external applications for support of legacy and third-party products.
- seamless integration with Oracle's middle tier web portal product, iPortal, and allows management of user information in an external directory, allowing integration with SSO technologies for other, non-Oracle applications.

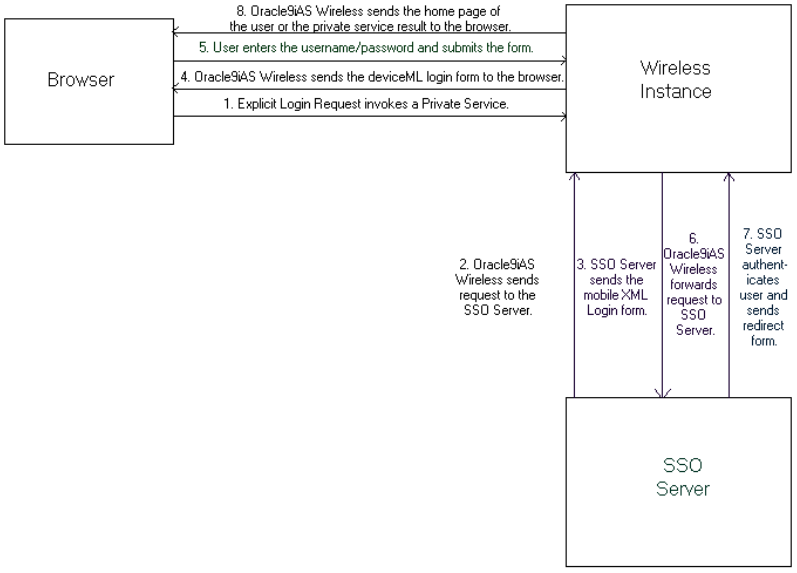
Users authenticate only once, and can access any SSO partner application. For example, a user authenticated by the Oracle9*i*AS Wireless server can access any SSO-enabled Partner Application (such as Oracle Portal) without authenticating again.

The following scenarios illustrate interactions between Oracle9*i*AS Wireless server and the SSO server.

10.2.1 Scenario 1: User Authentication by Oracle9*i*AS Wireless (device portal)

The Oracle9*i*AS Wireless server authenticates a user when the user sends an explicit Login Request (identified by URL parameter PAlogin=true), or tries to access a private service.

Figure 10–5 Interactions between Oracle9iAS Wireless and the Login Server



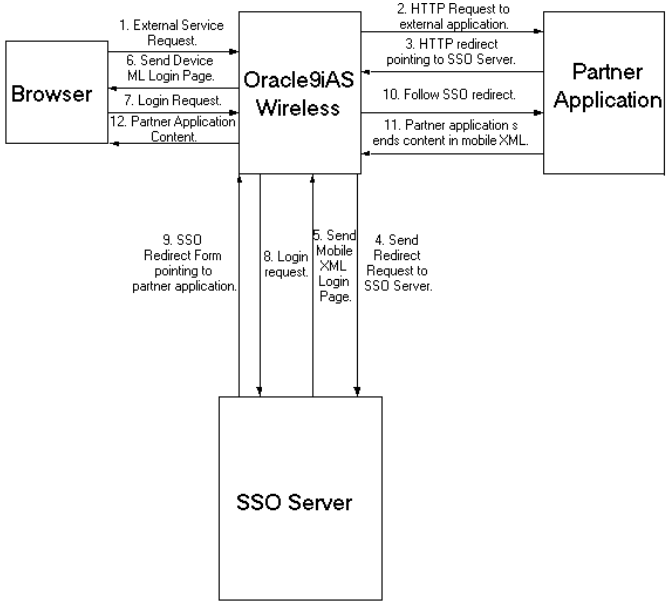
1. The user sends a Login Request or accesses a private service.
2. Oracle9*i*AS Wireless sends the Login request (without username / password) to the SSO Server.
3. The SSO Server checks the SSO cookie. If one is present, the login server identifies the user from the encrypted cookie and sends the SSO redirect form (step 7). This happens if the user is already authenticated by an external partner application (Section 10.2.2, "Scenario 2: User Authentication by an External Application"). If the SSO Cookie is not present, the SSO server sends the mobile xml login form to the Oracle9*i*AS Wireless server.
4. Oracle9*i*AS Wireless transforms the mobile xml login page to the appropriate device markup language and sends the device markup login page (such as WML) to the device browser.
5. The user enters the username / password and submits the Login Form.
6. Oracle9*i*AS Wireless forwards the Login Request (with user credentials) to the SSO Server.
7. SSO Server authenticates the user. If the authentication is successful, the SSO Server sends the SSO Redirect Form (if unsuccessful, the Login Form is sent [step 3 above]) to Oracle9*i*AS Wireless.

8. Oracle9*i*AS Wireless Server sends the home page of the user (or the private service result) to the device browser.

10.2.2 Scenario 2: User Authentication by an External Application

In Oracle9*i*AS Wireless-v902, the first request to the device portal (http://Oracle9*i*AS WirelessServer:port/ptg/rm) returns the home page of the anonymous user (Guest), or the home page of the identified *virtual User*. From that point, the user can access public services or can do an explicit login to access their private services. The unauthenticated user can execute HTTP Adapter-based public services, which points to an SSO-based partner application (such as Oracle Portal). The partner application may complete the SSO-based user authentication.

Figure 10–6 Interactions Between Oracle9iAS Wireless, Login Server and the External Application



1. An unauthenticated user executes an HTTP adapter-based service pointing to an SSO-based external application.
2. Oracle9*i*AS Wireless sends an HTTP request to the external application.
3. The partner application sends an HTTP Redirect pointing to the SSO Server.

4. Oracle9*i*AS Wireless follows the redirected URL.
5. SSO Server checks the SSO cookie. If one is present, the login server identifies the user from the encrypted cookie and sends the SSO redirect form (step 9 below). This happens if the user is an authenticated user. If the SSO Cookie is not present, the Login Server sends the mobile xml login form to the Oracle9*i*AS Wireless server.
6. Oracle9*i*AS Wireless transforms the mobile xml login page to the appropriate device markup language and sends the device markup login page (such as WML) to the device browser.
7. The user enters the username/password and submits the Login Form.
8. Oracle9*i*AS Wireless forwards the Login Request (with user credentials) to the SSO Server.
9. SSO Server authenticates the user. If the authentication is successful, SSO Server sends the SSO Redirect Form (if unsuccessful, the Login Form is sent [as in step 5 above]) to Oracle9*i*AS Wireless. After successful authentication, the Oracle9*i*AS Wireless session of the user is upgraded.
10. Oracle9*i*AS Wireless follows the SSO Redirect form. The redirect form points to the external partner application.
11. The partner application returns the service content in mobile XML.
12. Oracle9*i*AS Wireless transforms the mobile xml content to the appropriate device markup language, and sends the device markup content to the device browser.

10.2.3 Scenario 3: User Authentication by mod_osso

All Web-based Oracle9*i*AS Wireless applications (such as Customization) will authenticate users using mod_osso, which is a module plugged into Oracle HTTP Server. All of the Web-based Oracle9*i*AS Wireless applications running behind Oracle HTTP Server are treated as a single partner application. Users can access any of the applications after single sign-on.

The device portal uses the value of the HTTP header OssoUser_Guid to identify the mod_sso authenticated user.

> **Note:** When executing HTTP Adapter-based services pointing to external partner applications, the mod_sso authenticated user will have to be authenticated again. The reason for this is that for mod_sso authenticated users, the SSO cookies are stored in the PC browser.

10.2.4 Scenario 4: Voice based authentication

Voice authentication is accomplished by Oracle9*i*AS Wireless (locally) using the account number and the PIN of the user. Note that an authenticated user accessing external SSO partner applications from a voice device must re-authenticate (using username and password).

10.2.5 Global Logout

Oracle9*i*AS Wireless server participates in the SSO Global Logout. The following steps detail the interactions between Oracle9*i*AS Wireless, SSO Server and Partner Applications.

10.2.5.1 Scenario 1: Logout from Oracle9*i*AS Wireless

The user can click Oracle9*i*AS Wireless Logout to sign off.

1. The user sends a an Oracle9*i*AS Wireless Logout request (identified by URL parameter PAlogoff=true).
2. The Sign Off implementation of Oracle9*i*AS Wireless sends an HTTP request to the SSO Sign-Off URL.
3. The SSO server returns the mobile XML global logout page and a special HTTP header (X-Oracle-SSO-logout with value = true). The global logout page contains one image for each partner application that has the user session.
4. Oracle9*i*AS Wireless sends HTTP requests to each image link. This is done so that the user's session gets cleaned up in all the partner applications.
5. Oracle9*i*AS Wireless terminates the user's session.
6. If Logout is accomplished through Oracle9*i*AS Wireless link, then the home page of the "Guest" user is returned.

10.2.5.2 Scenario 2: Logout Link

The authenticated user can click on the logout link on the page returned by the SSO-based partner application. In this case, the logout link will point to the SSO sign-off URL.

1. The user clicks on the logout link which points to the SSO sign-off URL.
2. The SSO server returns the mobile XML global logout page and a special HTTP header (X-Oracle-SSO-logout with value = true). The global logout page contains one image for each partner application that has the user session.
3. Oracle9*i*AS Wireless sends HTTP requests to each image link. This is done so that the user's session gets cleaned up in all the partner applications.
4. Oracle9*i*AS Wireless terminates the user's session.
5. Oracle9*i*AS Wireless follows the done_URL of the global logout page.
6. The content returned by the done_URL is returned to the device.

10.2.5.3 Scenario 3: Logout from Web-based Oracle9*i*AS application

Since all Web-based Oracle9*i*AS applications are authenticated through mod_osso, and are treated as a single partner application, logout from any application triggers global sign-off and none of the applications will be accessible until the user signs on through mod_osso again.

10.2.6 Oracle9*i*AS Wireless-OID Integration

In this release, user information is stored centrally in OID. The SSO server uses an OID repository to authenticate users. The following table shows the attribute mapping between PanamaUser (stored in Oracle9*i*AS Wireless repository) and orclUserV2 user attributes (stored in OID).

Table 10–2 Attribute Mapping between PanamaUser and orclUserV2 user

PanamaUser	OID User
Name	orclcommonnicknameattribute (by default cn) specified in OID configuration
DisplayName	DisplayName
Enabled	orclIsEnabled
PasswordHint	orclPasswordHint
PasswordHintAnswer	orclPasswordHintAnswer

Table 10–2 Attribute Mapping between PanamaUser and orclUserV2 user

PanamaUser	OID User
Language and Country	preferredLanguage
TimeZone	TimeZone
DateofBirth	orclDateOfBirth
Globaluid	orclguid (orclguid attribute uniquely identifies OID Users)
Password	user password
Password Confirm	Confirms user password.
Gender	orcl header

*i*ASv902 administrators can use tools (such as Delegated Administrative Services [DAS]), to create a new User in OID or to modify attributes of an existing user. Alternatively, Oracle9*i*AS Wireless customers can implement their own user administrator tool to create / modify / delete users using Oracle9*i*AS Wireless model APIs.

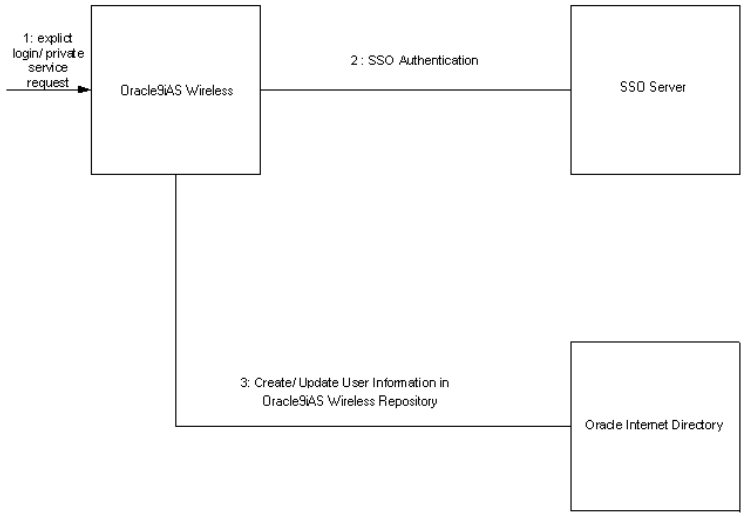
The user information is synchronized between Oracle9*i*AS Wireless and OID repositories using the following mechanisms:

- Oracle9*i*AS Wireless repository synchronization after user authentication
- PL / SQL based asynchronous synchronization
- Oracle9*i*AS Wireless model API interface

10.2.7 Oracle9*i*AS Wireless Repository Synchronization after User Authentication

Oracle9*i*AS Wireless synchronizes user information (stored in the Wireless repository) with OID after SSO authentication.

Figure 10–7 Interactions between Oracle9iAS Wireless, SSO and OID



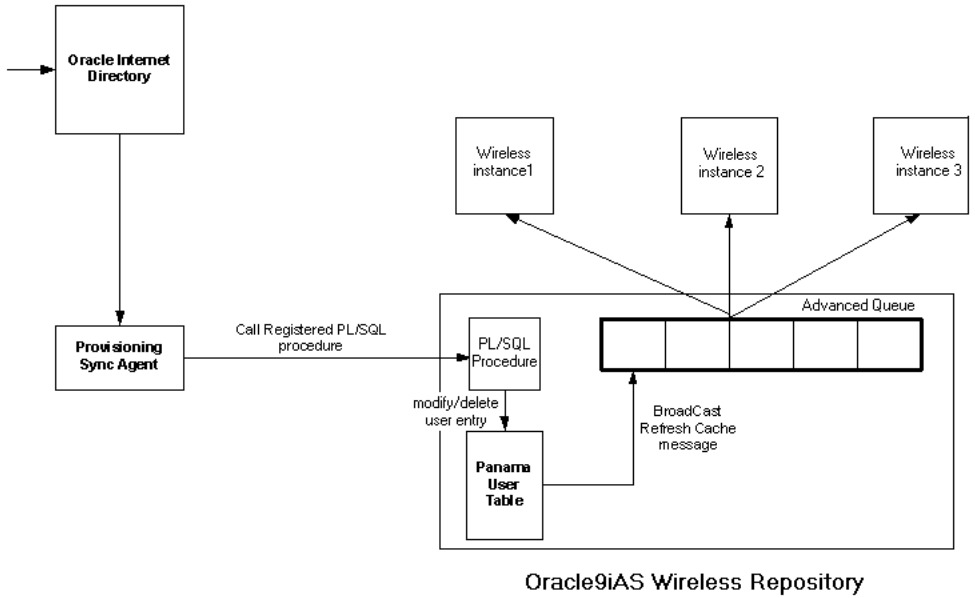
1. User sends an explicit login request or tries to access a private Service, or an external SSO partner application. The SSO server challenges user credentials and the user is authenticated.
2. If the authenticated user does not exist in the Oracle9*i*AS Wireless repository, Oracle9*i*AS Wireless retrieves the user information from OID and creates a new user in the Oracle9*i*AS Wireless repository. Otherwise, the User attributes in the local repository are synchronized with the attributes stored in the OID.

> **Note:** The reason for synchronizing User attributes with OID is that the PL/SQL notification mechanism does not guarantee real time notifications.

10.2.8 PL/SQL based asynchronous synchronization

The Oracle9*i*AS Wireless installation registers a PL/SQL procedure with OID. The PL/SQL procedure is invoked when a user is modified or deleted in OID.

Figure 10–8 Interactions between PL/SQL and OID



1. User attribute is modified, or the user is deleted in OID.
2. The Provisioning Synchronization agent picks up the modifications and calls the registered PL/SQL package.
3. The PL/SQL package accomplishes appropriate changes in the PanamaUser table (if required).
4. The trigger on the PanamaUser table broadcasts a RefreshCache message to all running instances of Oracle9*i*AS Wireless.
5. If the modified PanamaUser is cached by the running instances, the PanamaUser object is reloaded from the Oracle9*i*AS Wireless repository.

10.2.9 Oracle9*i*AS Wireless Programmatic Model API Interface

The ModelFactory.createUser() method creates a corresponding User in the OID repository.

The User.set methods update the corresponding User entry in OID for all the attributes. The following table shows the attribute mapping between PanamaUser (stored in Oracle9*i*AS Wireless repository) and orclUserV2 user attributes (stored in

OID). The User.delete() method removes the corresponding User from the OID repository. The current semantics of commit is preserved for the User modifications.

10.2.10 Oracle9*i*AS Wireless User Management Integrated with DAS

In Oracle9*i*AS Wireless integration mode, when you create a user through Webtool User Management, the request is first redirected to OID DAS (Delegated Administration Service), for entering Oracle9*i*AS User Common Attribute Values. After that, the request is redirected back to the Webtool User Management page for entering Wireless-specific attribute values.

The same applies for editing a registered Wireless user. The user is first edited through DAS and then through Webtool User Management.

10.2.11 WebCache Integration

Oracle9*i*AS Wireless is integrated with Oracle WebCache to improve page rendering performance and scalability. It must be clarified at the outset that WebCache is not deployed in the traditional sense with Oracle9*i*AS Wireless. WebCache is usually deployed in front of web-servers serving HTML content, and interacting with HTML clients and the web-server to cache dynamic content. However, with Oracle9*i*AS Wireless, the wireless runtime determines what content needs to be inserted into WebCache and when to expire content in the cache. WebCache, in this case, acts as a device adaptation cache rather than a reverse-proxy cache.

10.2.11.1 How Does this Work?

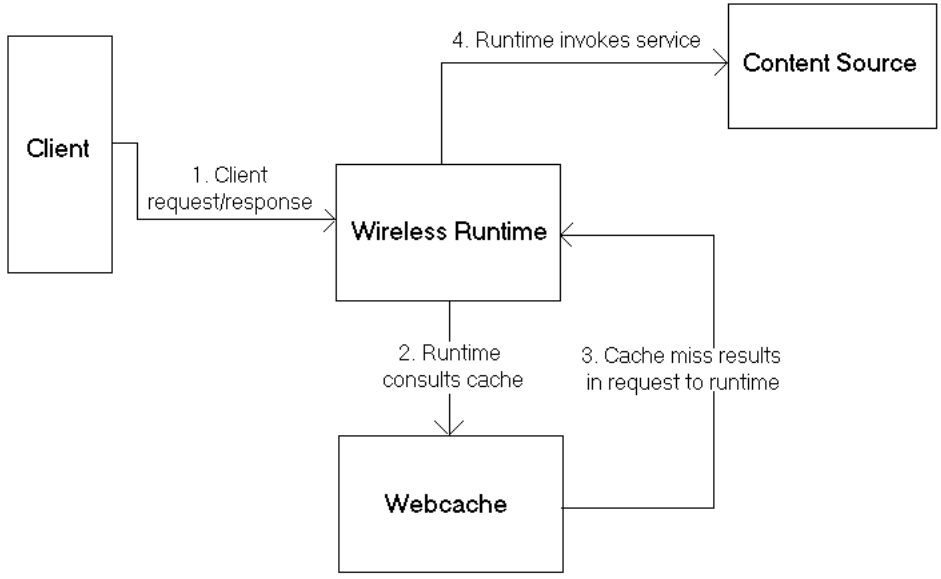
Since markup content is cached using WebCache, the performance and scalability benefits are due to two factors: reduced device adaptation costs, and significantly reduced adapter invocation costs. The savings in terms of device adaptation costs are due to the fact that content that can be shared across users and sessions is essentially transformed only once (per logical device) from its Mobile XML format. Secondly, since the content is not generated every time by an adapter, the total adapter invocation cost is significantly reduced for a site that has a large subset of cacheable pages.

10.2.11.2 A Cache Miss Scenario

1. An incoming request is received by the wireless runtime, which requests the cache for a page corresponding to the request and the device that made the request.

2. In this case, the page does not exist in the cache, causing WebCache to send a request back to the wireless runtime, requesting for the page.
3. This time, the runtime recognizes this request to be from WebCache, rather than from a client.
4. The runtime processes the requests following the traditional code-path of invoking the service corresponding to the request and transforming the content.
5. The transformed content is now returned as a response to the WebCache request.
6. WebCache examines the response to determine if the page is cacheable or not, and if it is, cacheable for what period of time.
7. Assuming that this particular page is cacheable, WebCache inserts the page into the cache with an expiration limit set to the page.
8. WebCache then serves this page out as a response to the original request from the runtime, which in turn uses this page as a response to the client request.

Figure 10–9 A Cache Miss Scenario

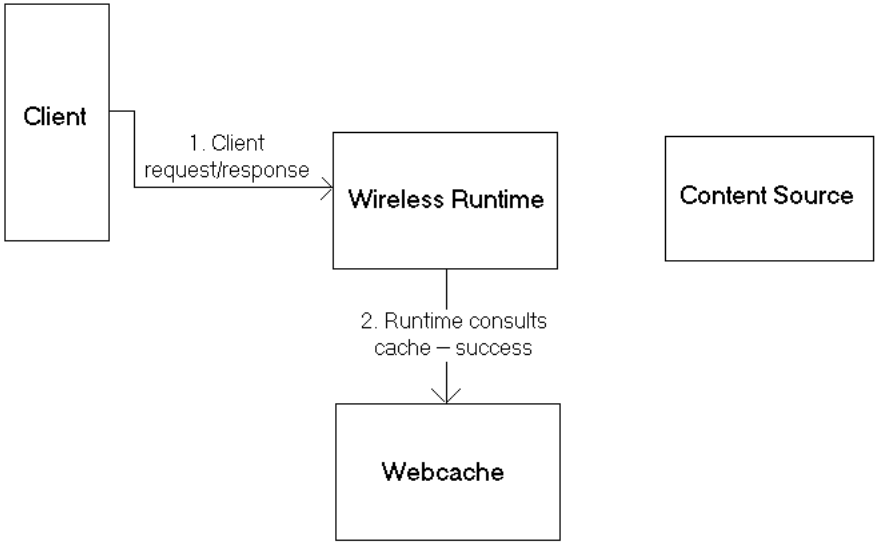


10.2.11.3 A Cache Hit Scenario

In this case, an incoming request from a client is for a page that has been cached by webcache.

1. The wireless runtime sends a request to webcache, which examines the cache to see if the page is cached or not.
2. If cached, it checks to see if the page has expired. If the page has not expired, it serves it out of the cache to the runtime, which in turn uses this page as a response to the client request.
3. However, if the page has expired, it once again follows the same routine as it would in the event of a cache miss.

Figure 10–10 Cache Hit Scenario



10.2.11.4 Configuration

10.2.11.4.1 Enabling Caching for the Site To cache dynamic content, it is necessary to enable WebCache in the first place. From the System Manager, click on the Site tab. Under the Administration section, in the Configuration sub-section, click on WebCache Configuration.

- To enable webcache, check the Enable WebCache checkbox.
- Next, enter the complete URL that corresponds to the webcache installation. Be sure to include the port number at which WebCache listens (default port is 1100) and the servlet path to the wireless runtime (default is /ptg/rm).
- Supply an invalidation password (default is Administrator). This should be the same as the WebCache invalidation password that is set from the WebCache administration console. See the WebCache Configuration Guide for details on how to perform this task.
- Provide an invalidation port (default is 4001). This should be the same as the invalidation port specified from the WebCache administration console. See the WebCache Configuration Guide for details on how to perform this task.
- Enter a timeout value for requests made to WebCache (default is 20 seconds). Ensure that this is at least 5 seconds less than the request timeout value from the WebCache administration console. See the WebCache Configuration Guide for details on how to perform this task.
- Click OK after the changes have been made.

10.2.11.4.2 Cache-enabling a Service The steps detailed above described how to enable caching for a site. For the cache to be of use, it is necessary to enable services to be cacheable.

- While creating a master service, the second step in the service creation wizard is the Caching step. To cache-enable a service, check the Cacheable checkbox. Once this is done, an Invalidation Frequency section appears. In this section, specify the frequency at which pages corresponding to the service must be removed from the cache.
- When a service is published from Content Manager, if the master service specified is cacheable, then the published service automatically becomes cacheable.

10.2.11.4.3 Invalidating Cache Content For any caching mechanism to be effective, it is necessary to perform invalidation of the cache contents at appropriate intervals. Invalidation of wireless content residing in webcache can be either policy-based or asynchronous.

Policy-based Invalidation—It is possible to specify in advance if a page should be cacheable or not. One of the ways to do this is by specifying the invalidation frequency of a service (as in the previous section). When a page is inserted into the

cache, the invalidation frequency of the service it belongs to is taken into account while determining how long the page should live in the cache.

It is also possible to dynamically specify the cacheability of a page. This is done at the content-source. If the page is to be specified as cacheable, the SimpleResult element should have a SimpleMeta child element. This element has a required attribute 'cache', which when set to 'yes', enables caching for the page and when set to 'no' disables caching. An optional attribute to be used in conjunction with a 'yes' value for the 'cache' attribute is 'ttl'. This can be used to specify in seconds the number of seconds the page should be cached before expiring it. For example:

```
<SimpleResult>
<SimpleMeta cache="no"/>
…..
</SimpleResult>
```

results in the page being non-cacheable, as below:

```
<SimpleResult>
<SimpleMeta cache="yes" ttl="300"/>
….
</SimpleResult>
```

results in the page being cached for 300 seconds.

Apart from using the SimpleMeta tag to specify cacheability, it is possible to use standard HTTP cache-control headers and ESI headers to specify cacheability for a page. Refer to your documentation on WebCache on how to specify cacheability using ESI headers.

The order in which cacheability for a given page is evaluated is as follows:

- Check for HTTP or ESI cacheability headers. These override SimpleMeta tags if any are present.
- SimpleMeta tags for a given page override the invalidation frequency for the service it belongs to.
- If neither the HTTP / ESI headers nor the SimpleMeta headers are present, the default cacheability policy for the service is applied to the page.

Asynchronous Invalidation—Despite specifying the cacheability policy for a page at the time of service creation or during the generation of the page, it may be necessary to explicitly invalidate content in the cache. It is possible to invalidate and refresh content in the cache based on a master service or a device.

From System Manager, click on the Site tab. Under the Administration section, in the Configuration sub-section, click on either 'Refresh webcache – Master service' or 'Refresh webcache – Device'.

- To invalidate all pages belonging to a master service, click on 'Refresh webcache – Master service', select a master service by clicking a radio button corresponding to the master service and click Refresh.
- To invalidate all pages with a given device markup, click on 'Refresh webcache – Device', select a device by clicking a radio button corresponding to the device and click Refresh.

10.2.11.5 Administration

If webcache is reinstalled on a different machine / port the WebCache settings must be reconfigured as detailed in the configuration section above.

If the wireless instance is reinstalled on a different machine, the location of the wireless instance should be modified in the 'Application Servers' of WebCache's administration console. See the WebCache Configuration Guide for details on how to perform this task.

10.2.11.6 Building a cacheable service

In this section we shall build a sample service that is cacheable using webcache. We shall also explore the means to control the cacheability of such a service dynamically.

The sample service displays the current time and therefore immediately demonstrates the cached status of the page. We follow the steps detailed below to create the service:

1. Create an external content source that can be invoked from an HTTP adapter. (As an aside, there is no requirement that a cacheable service need to be HTTP adapter based, any other adapter would do just as fine). We designate the content source as a simple JSP page which displays the current time in Mobile XML. For example:

```
<%@ page language="java" %>
<%@ page import="java.text.SimpleDateFormat"%>
<%@ page import="java.util.Date"%>

<%@ page session="false" %>
<%@ page contentType="text/html; charset=iso-8859-1" %>
        <SimpleResult>
        <SimpleContainer>
```

```
        <SimpleText>
        <SimpleTextItem>
        <%
                Date date = new Date();
                SimpleDateFormat formatter =
new SimpleDateFormat("yyyy.MM.dd G 'at' hh:mm:ss a zzz");
        %>
         <%=formatter.format(date)%>

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

Let us assume that this page is deployed at the URL:
http://mycontent-server.oracle.com/dateserv.jsp

2. We need to create a master service that uses this as the content source.
   - From the **System Manager** UI, clicking on the **Master Services** tab we create a new master service by clicking the **Create Master Service** button. In the mandatory fields (marked by an asterisk), we enter the value *Date Serv* for the **Name** of the master service and choose *HTTPAdapter* as the **Adapter** and ensure that the **Valid** checkbox is checked.
   - In the subsequent screen, we check the **Cacheable** checkbox and choose the **Invalidation Frequency** by specifying the **Cardinal** as *40* and **Unit** as *Seconds*, causing all pages corresponding to the service (in this case just one page) to be cached for 40 seconds.
   - In the next screen since our sample service does not have any **Init Parameters**, we click the **Next** button.
   - In the subsequent **Input Parameters** screen, we select the **URL** column and check the **Mandatory** field and enter the **Default Value** as the URL to our content source, i.e. http://mycontent-server.oracle.com/dateserv.jsp
   - We can skip the next couple of screens (by clicking **Next**) and create the master service by clicking the **Finish** button on the last screen.
3. Now, we need to publish the service from the **Content Manager**.
   - We click the **Add Service** button to add a link to the master service that was created earlier.

- We name the new service as *DateService* in the subsequent screen by entering *DateService* in the **Name** field. We also ensure that at least the **Visible** checkbox is checked and the Type is chosen as **Normal Service**.
- In the next screen we choose Date Serv as the master, drilling down to the folder it was created in and click on **Next**.
- We accept the default values in the next screen by clicking **Next**
- We publish the service by clicking **Submit** on the next screen.

4. We need to associate the service with an available **Group** for which we choose the Groups tab in the next screen.
   - We choose a **Group**, say *Guests* and click on the **Assign Services** button.
   - In the next screen under the list of Available Services, we choose *DateService* and click on the **Add To Group** button.
   - In the subsequent screen DateService should now be listed under **Group Accessible Services**. We click the **Finish** button to complete the service association.

The service is now accessible from the device portal. We can see that the time-stamp displayed as a result of invoking the DateService service does not change for 40 seconds, indicating that the service has been cached for 40 seconds and invalidated after. Please note that after a page in the cache has expired, the content is fetched from the content source only on a demand basis, i.e. after 40 seconds elapse Webcache will not refresh the content immediately, but will do so only after a new request for the page is received.

10.2.11.7 Dynamic specification of page invalidation

The time for which the cache can retain the page without refreshing it has been set to 40 seconds during the service creation. However, this value can be changed dynamically at the time of generation of the Mobile XML. This can be done in two ways:

10.2.11.8 Mobile XML markup

In this case the generated Mobile XML can have a SimpleMeta tag to attain this. Please see the Policy-base Invalidation sub-section in the previous section on how to do this. For our sample service, to ensure that the page is expired after 10 seconds (rather than the default of 40 seconds), the JSP page would be:

```
<%@ page language="java" %>
<%@ page import="java.text.SimpleDateFormat"%>
```

```
<%@ page import="java.util.Date"%>

<%@ page session="false" %>
<%@ page contentType="text/html; charset=iso-8859-1" %>
        <SimpleResult>
    <SimpleMeta cache=”yes” ttl=”300”/>
        <SimpleContainer>
        <SimpleText>
        <SimpleTextItem>
        <%
                Date date = new Date();
                SimpleDateFormat formatter =
new SimpleDateFormat("yyyy.MM.dd G 'at' hh:mm:ss a zzz");
        %>
         <%=formatter.format(date)%>

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

10.2.11.9 ESI headers

Responses from the content source may contain ESI headers as part of HTTP headers that can dictate cache expiration behavior. Using ESI headers entail no changes to the Mobile XML. The following ESI header expires the page is 30 seconds.

```
Surrogate-Control: max-age=30+60, content="ESI/1.0"
```

For more information on ESI headers, please refer to the Webcache Developer’s Guide.

10.2.12 Oracle Portal and Oracle9*i*AS Wireless

Oracle9*i*AS Portal is a web-based application model for building and deploying e-business portals. It provides a environment for accessing and interacting with enterprise software services and information resources. Portal provides a framework that integrates web-based resources such as web pages, applications, business intelligence reports, and syndicated content feeds, within standardized, reusable information components called portlets.

A portlet is an area of HTML / XML located within a defined area of a Web page. Portlets communicate with the portal through an entity called a provider. Portlets form the fundamental building blocks of a Oracle9*i*AS Portal page. Each portal page

consists of content presented through one or more portlets and links that allow the user to navigate to another page to take some action.

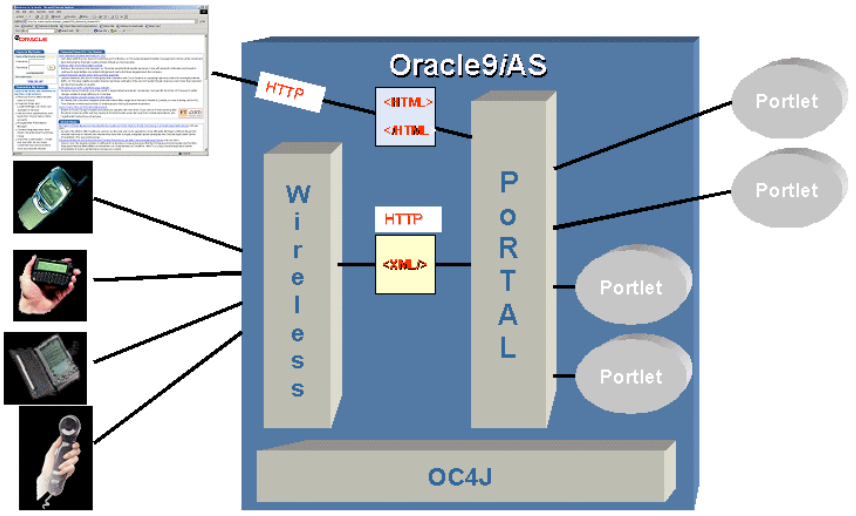
Portlets summarize, promote or provide basic access to an information resource. The portlets allow information resources to be personalized and managed as a service of Oracle9*i*AS Portal. The portal framework provides additional services including single sign-on, content classification, enterprise search, directory integration, and access control. OraclePortal traditionally has been supporting Desktop / PC Web browsers. Starting in Orcale9*i*AS releas2.0 OraclePortal, besides support standard web browsers, will enable Oracle9*i*AS Portal pages to be accessed from wireless devices. OraclePortal, working in conjunction with Oracle9*i*AS Wireless, automatically transforms the portal page structure that is appropriate for the wireless devices. Portal generates the Page structure in Oracle9*i*AS Wireless XML, for all request from wireless device, and rendered to the device by Oracle9*i*AS Wireless. This allows portlets to provide wireless interface using OraclePortal, through Oracle9*i*AS Wireless.

10.2.13 Oracle Portal as a Wireless Service

To enable Oracle9*i*AS Wireless access to Portal, the Portal must be deployed as a Wireless service in the Oracle9*i*AS Wireless repository. Each Portal installation is deployed as an HTTP Adapter service in Oracle9*i*AS Wireless. Multiple Portals may be deployed on a single Wireless instance. The HTTP adapter service accepts a URL as a configuration parameter and must be set to the URL of the Portal's home page. To create a Wireless service, a Master Service definition based on an HTTP adapter must be created using the Oracle9*i*AS Wireless Webtool. Also, you must create an OraclePortal Service based on the HTTP adapter Master Service.

OraclePortal redirects requests from a Wireless device to an Oracle9*i*AS Wireless server. The Oracle9*i*AS Wireless Server accepts the request and invokes the OraclePortal home page over HTTP and accepts the response generated (in Oracle9*i*AS Wireless XML), from OraclePortal. The XML response, generated by OraclePortal, is then adapted to the native device markup by the Oracle9*i*AS Wireless server. All further requests and responses between Wireless device and OraclePortal is mediated by the Oracle9*i*AS Wireless Server.

Figure 10–11 Oracle Portal Integration



Wireless devices make the first request to OraclePortal server and Portal redirects the device request to Wireless Server. The Portal appends two parameters to the redirected URL, the two query parameters appended are "PAoid" and "PAhome". Both PAoid and PAhome contain the value of the object id (service-id in the Wireless repository) of the Portal's HTTP adapter service. The syntax of the redirected URL is:

```
http://9iASWSerrver:port/ptg/rm?PAoid=<OraclePortal object
id>&PAhome=<OraclePortal object id>
```

The PAoid parameter allows the Wireless server to directly launch the Portal home page, without having to navigate through the Wireless server's folder and service hierarchy. The PAhome sets the Portals Home Page as the home page for the current wireless session.

10.2.14 Developing Wireless Portlets

Portlets are owned by entities called Providers, and one Provider can manage one or many portlets. Providers are the backbone behind the Portlets being displayed on each page. Portal supports a Web Provider framework that is written as a web application. It is installed and hosted on a web server and is remote from the Portal.

A portlet exposed as a Web Provider can be developed in any web language. A Web Provider communicates with Oracle9*i*AS Portal using SOAP(XML).

OraclePortal supports a Java based Portal Developer Kit (PDK) framework to develop portlets and services. The Java PDK Framework is a set of services that enable Java programmers to easily create portlets from existing Java-based applications (Java, Java Servlets, and JSPs). It provides an abstraction to handle communication with Oracle9*i*AS Portal, default classes to simplify portlet creation, and exposes APIs for end-user customization, session storage, security, and logging.

For Wireless devices, OraclePortal will support Portlets that generate Oracle9*i*AS Wireless XML. To enable wireless access Portlets must generate Oracle9*i*AS Wireless XML and indicate such capability using the Java PDK framework. The Java PDK framework uses a Provider.xml file to discover the capabilities of the Portlets supported by a Provider. Refer to OraclePortal's PDK-Java User's Guide for more information.

Following is a overview of tags (in the Provider.xml file) that indicates the wireless capabilities of a Portlet.

```
1.<acceptContentType>
          Usage:
<acceptContentType>text/vnd.oracle.mobilexml</acceptContentType>
```

This value "text/vnd.oracle.mobilexml" indicates that the portlet is capable of generating Oracle9*i*AS Wireless XML required for Wireless access. A portlet can be enabled for both HTML (PC Desktop) and Wireless Access by indicating it can accept both the content types such as:

```
<acceptContentType>text/vnd.oracle.mobilexml</acceptContentType>
           <acceptContentType>text/html</acceptContentType>
```

If the Portlet is capable of generating only Oracle9*i*AS Wireless XML (text/vnd.oracle.mobilexml), then (unless otherwise indicated) the Portlet will transform the Oracle9*i*AS Wireless XML to HTML for PC Desktop clients.

```
2.<mobileFlags>
          Usage: <mobileFlags>MOBILE_ONLY</mobileFlags>
```

Portlets can set this value to MOBILE_ONLY and hence indicate that this Portlet must be rendered in wireless devices only. This will prevent the default behavior of a Portal to transform Oracle9*i*AS Wireless XML, generated by the Portlet and rendered to PC Desktop clients.

```
3.<showLink>
          Usage:<showLink>true</showLink>
```

Portal renders all the Portlets on Wireless devices as links. Portlets must set this value to True to be rendered on a wireless device. A value of True allows the Portal to generate a Link, pointing to the Portlet content, on the wireless device.

```
4.<linkPage>
           Usage:<linkPage
class="oracle.portal.provider.v2.render.http.ResourceRenderer">
                       <resourcePath>/mypath/mypage.jsp</resourcePath>
                       <contentType>text/vnd.oracle.mobilexml</contentType>
                       </linkPage>
```

This tags holds the pointer to the resource which generates the required link that is rendered on a wireless device. This resource must generate Oracle9*i*AS Wireless XML. Below is a sample link page implemented in JSP.

```
<%@ page session="false" contentType="text/vnd.oracle.mobilexml" %>
    <SimpleHref target="/mypath/mywireless.jsp" label="Go">
          Wireless HelloWorld
    </SimpleHref>
```

The new version JPDK has been updated to understand these wireless properties of a Portlet. The JPDK also supports wireless specific request information like location and device information, which can be accessed by the Portlets through the JPDK APIs.

10.2.15 OraclePortal, Oracle9*i*AS Wireless and Single SignOn (SSO)

Both OraclePortal and Oracle9*i*AS Wireless depend on Oracle's SSO solution for user authentication and login. This integration allows the user to invoke protected applications defined on both systems and eliminates multiple login dialog boxes for users.

Oracle9*i*AS Wireless Server upgrades the session context of a user to an "authenticated" state when any service or application (HTTP Adapter services) validates the user credentials with the SSO server. When OraclePortal, mobile application, validates the credentials of a user with the SSO Server, the session context in Oracle9*i*AS Wireless is also updated. This allows wireless Portlets deployed on OraclePortal to uses services such as User Location Picker, Routing, Mobile Positioning supported by the Oracle9*i*AS Wireless Server.

10.2.16 Portlets for Services Deployed on Wireless Server

You can use OraclePortal's services to provide a PC Desktop view of your Oracle9*i*AS Wireless services. You can use Portal's JPDK framework to provide a "showPage" and "editPage", for web-based customizations.

Since the Portal itself can be accessed from a wireless device, you must also provide a mobile Portlet. On a wireless device, the mobile Portlets are rendered as links and can be made to point to a service deployed on the Oracle9*i*AS Wireless server. You can use Portal's JPDK framework to provide a "linkPage" that generates the appropriate link for your wireless service. To point to a wireless service from a mobile portlet you can use following URL syntax in your Oracle9*i*AS Wireless XML:

```
target="___REQUEST_NAME__?___SESSION__&amp;PAoid=<PAoid of Wireless Service>"
```

The Wireless server will replace all "___<Name>__" to the correct values at runtime and will invoke a service define in the Oracle9*i*AS Wireless repository.

The following is a sample link page:

```
<%@ page session="false" contentType="text/vnd.oracle.mobilexml" %>
          <SimpleHref target="/___REQUEST_NAME__?PAoid="+PAoid + "&amp;___
SESSION__" label="Go">
                My Wireless Service
          </SimpleHref>
```

Mobile devices make the first request to OraclePortal server. Portal redirects the device request to Oracle9*i*AS Wireless Server, over HTTP, and appends two parameters to the redirected URL. The two query parameters are "PAoid" and "PAhome". Both PAoid and PAhome contain the Portal's object/service id. The typical syntax of the redirected URL are:

http://Oracle9*i*AS WirelessSerrver:port/ptg/rm?PAoid=<OraclePortalServiceid>&PAhome=<OraclePortalService id>

The PAoid parameter allows the Wireless server to directly launch the Portal home page, without having to navigate through the Wireless server's folder and service hierarchy. The PAhome sets the Portals Home Page as the home page for the current wireless session.

10.2.16.1 Webtool and Customization as Portal Providers

The post-installer automatically registers Webtool and Customization as two Oracle Portal Providers. Thus, if an Oracle Portal user selects the two providers he/she will see two portlets: one for Webtool, and one for Customization. If the URL for

Webtool or Customization is changed, the provider can be registered from Wireless System Manager, part of Oracle Enterprise Manager. For more information, see *Oracle9i*AS Wireless *Getting Started and System Guide.*

10.3 Wireless Services

10.3.1 Wireless Services Overview

Services enable end users to access the functionality of Oracle9*i*AS Wireless. They represent the link between the content source and the delivery target. Services tie a specific data source (through an adapter) to the different devices.

There are different types of services:

- **MasterService**—provides the actual implementation of the service. MasterServices specify the adapter used for the service and any service-specific parameters.
- **Link**—a pointer to a service. In most cases Links are used to publish MasterServices to end users and to customize the MasterService parameters.
- **Module**—a pointer to a MasterService with a known URL.
- **Folder**—container for other services, including other Folders. Used to build service trees.
- **ExternalLink**—a service that points to an external resource.

10.3.1.1 MasterService

MasterServices provide the basic wireless functionality. They are the actual implementation of the service. Each MasterService is based on one adapter. A MasterService sets values for the adapter init, input and output parameters. Each MasterService creates its own instance of the adapter it uses. Therefore, several services can use the same type of adapter, and each can pass its own service-specific argument values.

It is recommended that you build all MasterServices using the HTTPAdapter. That gives you the flexibility to implement the service business logic using JSPs or other web technologies.

10.3.1.2 Link

Links are used to further customize existing services by overriding the values of their parameters.

When a Link service is invoked the Wireless server will merge the parameters with the parameters of the service the Link points to, and invoke that service.

Links are also used to better organize services into user service trees. They give you the flexibility to publish the same service under different names and in different folders (different levels in the service tree). If you do not override any parameter values, then invoking the link is the same as invoking the service it points to.

10.3.1.3 Module

Modules are wireless services with well-known virtual URL (OMP URL, that is, omp://my.module).

Modules can be called from any application or module and may be instructed to return control to another application or module. Calls may be nested to any level. This mechanism of bi-directional linking allows quick applications assembly.

An important difference between a module and a regular service is that the module receives information about the service it needs to return to after it is done. This is not always the caller of the module (the module caller may want the module to return to a different service).

10.3.1.4 Folder

Folders are containers for other services. They are used to better organize user-accessible services into a service tree. The content of a folder is displayed by invoking its rendering service—a special service associated with each folder.

The system rendering service displays the folder child services ordered by the specified sort rule.

Optionally, you can specify icons and audio files to be displayed/played when a service link is displayed in the folder content or when the service is invoked.

10.3.1.5 ExternalLink

An ExternalLink is a wireless service that points to an external resource. The external resource is typically a Web page that serves content in a format supported by the target device.

Oracle9*i*AS Wireless does not process the content of the ExternalLink target. As a result, ExternalLink services are not available to all targeted devices, as are other Wireless services. In most cases, ExternalLinks are set in the Customization portal by the end user, not in the Service Designer.

10.3.2 Access Control

There are two type of services in terms of accessibility:

- **User Private Services**—accessible by a single user.
- **Shared Services**—accessible by multiple users.

There are different rules that apply to those two type of services.

The user private services are services that reside in the user home service tree. The user can access all of those services. No other user can access those services.

The shared services in contrast are accessed by multiple users. The access is controlled by the User - Group - Service relationship. When you assign a service to a group, all users from that group can access the service.

10.4 Device and Network Adaptation

This section describes how to create and manage Oracle9*i*AS Wireless transformers.

10.4.1 Logical Device

Logical Device in Oracle9*i*AS Wireless represents either a physical device, such as an Ericsson mobile phone or an abstract device, such as ASYNC. The Logical Device stores the attributes of the physical device/ browser and device transformers. The Oracle9*i*AS Wireless server uses the device transformer of the Logical Device associated with the request to transform mobile xml service result to device mark up language.

Each request in Oracle9*i*AS Wireless is associated with a Logical Device. The Device Detection process, i.e. finding out the Logical device corresponding to a request, is done for each Oracle9*i*AS Wireless request. Device Detection mechanism is discussed later in the chapter.

The following table lists the Logical Device attributes. These attributes can be retrieved and modified using programmatic java api's. Refer to the javadoc of oracle.panama.model.Device interface.

Table 10–3 Logical Device attributes

Attribute Name	Description
Name	Name of the logical Device
Description	Description of the Logical Device

Table 10–3 Logical Device attributes

Attribute Name	Description
Encoding	The Character Encoding to be used by the Device. This attribute specifies the Character encoding used by the device browser to send URL parameters. Also the content returned in response to a request is encoded using the encoding of the logical device.
Preferred Mime Type	mime type supported by the device, for example text / html for devices supporting HTML

The above mentioned attributes can be used by the Transformers, Adapters, Folder Renderer hooks or external Http Adapter based Services to generate custom content for the device. The Logical Device attributes are passed to the external Http Adapter based services through HTTP headers. See Section 10.7, "Adapters" for more information.

Oracle9*i*AS Wireless server is shipped with pre-built Logical Devices. Customers can add additional logical devices or can modify existing Logical Devices if any of their physical devices can not be mapped to an existing Oracle9*i*AS Wireless Logical Device. Refer to *Oracle9iAS Wireless Getting Started and System Guide* for details on how to add or modify Logical Devices.

10.4.2 Device Detection

The Device Detection in Oracle9*i*AS Wireless can be customized by specifying a hook class that implements the interface **oracle.panama.rt.hook.DeviceIdentificationHook.** The default implementation of the hook is provided in **oracle.panma.rt.hook.DeviceIdentificationPolicy** class.

The default (built-in) implementation uses the User-Agent String to Logical device mappings, stored in the LogicalDevice model object, to identify the logical device from the request. Note that in previous releases of Oracle9*i*AS Wireless the User-Agent String to logical device mapping was specified in **oracle/panama/core/admin/UserAgents** properties file.

The User-Agent string can contain wild card character '*'. For example, the User-Agent String '*DS*' will match all the User-Agent values containing 'DS'.

The Device Detection algorithm:

1. Match the User-Agent http header value with all the User-Agent Strings. If there is a match then use the Logical Device corresponding to the matched User-Agent String, else go to step 2. In case of multiple matches the Logical

Device corresponding to the User-Agent string with maximum number of non-wild card characters is used.

2. Find all the Logical Devices whose mime type attribute matches the value of the Accept http header. Go to step 3.
3. If the request contains x-up-devcap-screenpixels and x-up-devcap-screenchars http headers then, find the closest matching logical device using ScreenWidth, ScreenHeight, ScreenRows, ScreenColumns attributes of the Logical Device. Else select any logical device.

10.4.3 Image Support

The devices and browsers available in the market today support different image formats, for example, WML devices support wbmp image formats whereas Palm supports gray scale depth 2 image formats. The "image Format Preferences" attribute of Logical device stores all the image mimetype and corresponding file extension supported by the device. This attribute of Logical Device is used by the Transformers to transform the <SimpleImage> mobile xml element.

The mobile xml developer can use the "available" attribute of <SimpleImage> element to specify the list of image file extensions available. The transformer appends the file extension, supported by the device, to the "src" attribute of the <SimpleImage> element. The "src" attribute of <SimpleResult> specifies the location of the image file.

For example, the following <SimpleOracle9*i*AS Wireless element> specifies that the image_file is available in "gif", "wbmp"and "g2gif" formats.

```
<SimpleImage src="http://IASWServer:port/image_file" available="gif wbmp g2.gif"
/>
```

For devices supporting only g2.gif extension the above <SimpleImage> will get transformed to:

```
<img src="http://IASWServer:port/image_file.g2.gif">
```

10.4.4 Transformer

Oracle9*i*AS Wireless supports Device Transformers.

The Device Transformers transform mobile xml document to the device markup language. The transformation logic can be implemented in an XSL stylesheet or in Java.

The Result transformers convert content from AdapterResult format to Mobile XML 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 service. A chained service is an Oracle9*i*AS Wireless service that invokes another service. Result Transformers are deprecated in Oracle9*i*AS Wireless 9.0.2 version.

The following table lists the attributes stored in the Device Transformer objects. These attributes can be accesses by java programmatic apis' – refer to javadoc of oracle.panama.model.Transformer, oracle.panama.model.JavaTransformer and oracle.panama.model.XSLTransformer interface.

Table 10–4 Device Transformer objects attributes

Attribute Name	Description
Name	Name of the transformer
Mime Type	The mime type of the target device markup language. For example, text / html
Mobile XML DTD version	The mobile xml dtd version supported by the transformer.
XSL Stylesheet	The XSL Stylesheet implementing the transformation logic. This attribute is valid only for XSL based Transformers
Java Class	The class path of the class implementing the transformation logic. This attribute is valid only for Java based Transformers.

10.4.4.1 Java Transformers

Transformers can implement transformation logic in Java by implementing *oracle.panama.rt.xform.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.
 *
```

```
 * @since     Oracle9i Application Server Wireless Edition
 */
public interface RtTransformer {

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

}
```

Oracle9*i*AS Wireless run time calls the transform method of the Java transformer to transform the mobile xml document to device markup. The parameter *element* contains the ServiceContext, and the device markup result is written to the *out* parameter of the method. The ServiceContext contains the input parameters of the request, attributes of the logical device corresponding to the request and SimpleResult (mobile xml document).

The class implementing *oracle.panama.rt.xform.RtTransformer* interface must provide a default constructor (that is, constructor without arguments) and the *transform()* method should be thread-safe.

The ServiceContext Element passed to the *transform()* method of *RtTransformer* class is of the form.

```
<ServiceRequest>
   <Arguments>
       <Inputs>
```

All the input arguments passed to the Service – this includes the service arguments and other arguments listed below

```
……..
       </Inputs>
   </Arguments>
   <Result>
The mobile xml content
…….
   <Result>
</ServiceRequest>
```

The input argument is of the form

```
<name ….>value</name>
```

where “name” is the name of the input argument and “value” is the value of the input argument.

The following table lists the input arguments other than the service input arguments, which are passed to the Service.

Table 10–5 Input arguments

Name of the Input Argument	Description
_LOGICAL_DEVICE	Name of the logical device corresponding to the request.
_ScreenColumns	The value of screen columns attribute of the logical device corresponding to the request
_ScreenRows	The value of screen rows attribute of the logical device corresponding to the request
_ScreenWidth	The value of screen width attribute of the logical device corresponding to the request
_ScreenHeight	The value of screen height attribute of the logical device corresponding to the request
_DeviceCategory	The value of device category attribute of the logical device corresponding to the request
_SoftKeys	The value of soft keys attribute of the logical device corresponding to the request
_MaxDocSize	The value of max doc size attribute of the logical device corresponding to the request
_ImagePreferences	The value of image preferences attribute of the logical device corresponding to the request
_User	The name of the User.
_UserLanguage	The language preference of the User
_FirstAcceptLanguage	The first language specified in the Accept-Language HTTP header.
USER-AGENT	The User-Agent HTTP header value of the request Note: All the HTTP headers of the Oracle9*i*AS Wireless request are added to the Service Context
COOKIE	The Cookie HTTP header value of the request
CONNECTION	The Connection HTTP header value of the request

Table 10–5 Input arguments

Name of the Input Argument	Description
ACCEPT	The Accept HTTP header value of the request
HOST	The Host HTTP header value of the request
REFERER	The Referer HTTP header value of the request
ACCEPT-LANGUAGE	The Accept-Language HTTP header value of the request
ACCEPT-ENCODING	The Accept-Encoding HTTP header value of the request
_SERVICE_NAME	The name of the invoked Service.
_SERVICE_NAME_ENC	The URL encoded name of the invoked Service.
_SERVICE_URL	The URL of the invoked Service
_SERVICE_URL_ENC	The URL encoded value of the URL of the invoked Service
PAoid	The object id of the invoked service
_REQUEST_NAME	The path component of the Servlet. For e.g. /ptg/rm
_HTTP_REQUEST_NAME	The URL path of the Servlet. For e.g. http://IasServer:Port:7777/ptg/rm
_ABS_REQUEST_NAME	The URL path of the Servlet. For e.g. http://IasServer:Port:7777/ptg/rm
_HTTPS_REQUEST_ NAME	The HTTPS URL path of the Servlet. For e.g. https://IasServer:Port:7778/ptg/rm
PAsid	The session id of the request. For e.g. 100BoNrXdG
_SESSION	The session id name/value pair. For e.g PAsid=100BoNrXdG
PAservlet	The name of the Servlet. For e.g. rm
AMP_EXPLICIT	

10.4.5 XSLT Transformers

Device transformer logic can be implemented in XSL stylesheet. XSL stylesheets are XML documents that specifies the processing rules for other XML documents. An XSLT stylesheet, like java transformers, is written for a particular mobile XML DTD. When it finds the element in source document, it follows the rules defined for the element to format its content. The ServiceContext element is passed as the source document to the stylesheet.

10.4.5.1 Creating XSL Transformer

In this section we will implement a very simple XSL stylesheet which handles only <SimpleTable> mobile xml element. It uses the value of screen width attribute of the logical device, passed as _ScreenWidth element in the ServiceContext, as the width of the generated HTML table. The Stylesheet converts source mobile xml documents to HTML.

Example mobile xml document handled by our custom stylesheet.

```
<SimpleResult>
   <SimpleContainer>
      <SimpleTable>
         <SimpleRow>
            <SimpleCol>Row1 column1</SimpleCol>
            <SimpleCol>Row1 column2</SimpleCol>
         </SimpleRow>
         <SimpleRow>
            <SimpleCol>row2 column1</SimpleCol>
            <SimpleCol>row2 column2 </SimpleCol>
         </SimpleRow>
      </SimpleTable>
   </SimpleContainer>
</SimpleResult>
```

The stylesheet implementation.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:p2g="http://www.oracle.com/XSL/Transform/java/oracle.panama.core.xform.XSJ
ava" exclude-result-prefixes="p2g">
This template matches the root of the document.
   <xsl:template match="/">
      <xsl:apply-templates select="//SimpleResult"/>
   </xsl:template>
```

Template for SimpleResult element.

```
   <xsl:template match="SimpleResult">
      <HTML>
         <BODY>
            <xsl:apply-templates select="SimpleContainer/SimpleTable" />
         </BODY>
      </HTML>
   </xsl:template>
```

Template for <SimpleTable> element.

```
   <xsl:template match="SimpleTable">
      <TABLE>
```

The width attribute of the table is set to the value of _ScreenWidth element.

```
         <xsl:attribute name="width">
            <xsl:value-of select="//_ScreenWidth" />
         </xsl:attribute>
         <xsl:for-each select="./SimpleRow">
            <xsl:apply-templates select="."/>
         </xsl:for-each>
      </TABLE>
</xsl:template>
```

Template for SimpleRow element

```
   <xsl:template match="SimpleRow">
      <TR>
         <xsl:for-each select="./SimpleCol">
            <xsl:apply-templates select="."/>
         </xsl:for-each>
      </TR>
   </xsl:template>
```

Template for SimpleCol element.

```
   <xsl:template match="SimpleCol">
      <TD>
         <xsl:value-of select="." />
      </TD>
   </xsl:template>
</xsl:stylesheet>
```

10.4.5.2 Transformer Version

Oracle9*i*AS Wireless server supports multiple transformers, one for each version of the mobile XML DTD, for Logical Devices. The run time selects the transformer depending on the DTD version specified in the mobile XML Document. All the mobile xml documents should include the following DOCTYPE declaration

<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult x.y.z//EN" "http://xmlns.oracle.com/ias/dtds/SimpleResultxyz.dtd">

The DTD version is specified by the string "x.y.z", where x is incremented for every major revision of DTD, y is incremented for minor revisions, and z can be incremented by the customer for customer specific DTD enhancements.

For backward compatibility all the mobile xml documents, which do not contain the DOCTYPE declaration, will be assumed to be confirming to 1.0.0 version (i.e pre 9.0.2 version of the DTD)

The algorithm to find the transformer for a mobile xml document confirming with DTD version x.y.z

1. Find all transformers with major number "x". If no transformers are found then log an error message and return.
2. From the transformer set returned from step1, find all the transformers with minor number 'y'. If a transformer is found then go to step4
3. From the transformer set (returned from step1) choose the transformer with minimum minor number which is greater than y. If no such transformer exists then choose a transformer with maximum minor number which is less than y.
4. From the transformer set returned from step2, choose the transformer with customer version number 'z'. If no such transformer is found then choose the transformer with minimum customer version number which is greater than z. If no such transformer exists then choose a transformer with maximum customer version number which is less than z.

The following HelloWorld.xml confirms to DTD version 1.1.0 (which is the current version of the DTD).

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1.0//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
<SimpleContainer>
  <SimpleText>
    <SimpleTextItem>Hello World </SimpleTextItem>
  </SimpleText>
</SimpleContainer>
</SimpleResult>
```

Note: For information on using Service Designer to create and manage Transformers, and for all Service Designer functions, see "Managing Transformers" in *Oracle9iAS Wireless Getting Started and System Guide*.

10.5 Asynchronous Server

10.5.1 Asynchronous Server Architecture

Oracle9*i*AS Wireless presents a framework to develop mobile applications to be accessed "from anywhere, from any device, using any protocol". The Asynchronous Server Kernel, also known as ASK makes the "any protocol" promise possible.

Conventionally, the entry point into an application server is through the HTTP protocol. This limits applications built on an application server to only clients with Web capability. This server restriction is a problem for mobile market users because the vast majority of mobile users do not have, or are not enabled with Web access. These users are almost certain to have some kind of message capabilities though (such as e-mail, SMS, etc.). Now the dilemma is whether one should build applications for such users specifically, depending on their capability, or ignore them because the application server just cannot deal with the mobile market.

With Oracle9*i*AS Wireless, the dilemma is solved for developers, without them doing anything at all. With the introduction of ASK, mobile applications can be accessed through the usual HTTP protocol, as well as any other messaging protocols (such as e-mail, SMS, etc.). Developers can focus on building their application logic, and Oracle9*i*AS Wireless will do the proper connection, session management, and interpretation of user requests. A mobile service is invoked the same way regardless of which protocol handles the incoming requests, offering complete transparency to application developers to allow access to their services.

10.5.2 Key Technical Challenges

10.5.2.1 Multiple transport protocol support

One of the most obvious challenges is to be able to support multiple protocols. It is not desirable to build the same functionality to work with e-mail, then SMS, then some other protocols. Oracle9*i*AS Wireless offers access to the same application regardless of the protocol used by clients. Hence the immediate challenge is to be able to support multiple protocols uniformly.

10.5.2.2 The asynchronous nature of messaging protocols

In contrast to the HTTP protocol, (commonly referred to as the synchronous protocol) messaging protocols such as SMS or e-mail are asynchronous. It is asynchronous because unlike HTTP, they are not based on a "request and response" model. A single atomic operation is typically one way. For example, when you use a Web browser, you enter a URL and make the request, then you wait for the result to come back. In messaging protocols (such as SMS) sending a message itself completes one operation. Most applications respond to user requests so HTTP is usually adequate. To enable the same application be accessed through asynchronous protocols presents a challenge on how such behavior can be mimicked with protocols such as SMS or e-mail.

10.5.2.3 Supporting Sessions

Another big challenge is that most applications are session based; multiple requests and responses are typically required to complete a task. Applications are able to maintain sessions in the Web world because the client, a Web browser, has built in capabilities such as cookies to facilitate session semantics. This is not the case for an e-mail or SMS client. They do not have any such ability built in to support conversational applications.

10.5.2.4 User Navigation

A Web browser offers a User Interface for navigating through applications (examples include clicking on a hyperlink and traversing through a menu or a series of steps to complete certain functionality). Clients that work with other protocols such as SMS or e-mail typically do not have similar navigation power. The challenge here is to offer similar navigating capability to such clients so that applications can be independent of the protocols.

10.5.2.5 Naming/Addressing an Application

In the Web world, applications are typically assigned a URL. The URL is how the application can be identified and requested. Clients for messaging are typically plain text devices; there is no convention on how to name a service, but consistency across protocols is needed.

10.5.3 Technical Solutions and Features

ASK combines functionality of a HTTP server and portions of a Web browser to provide its functionality.

10.5.3.1 Multiple transport protocol support

This challenge is a relatively easy one. Built on top of the Oracle9*i*AS Wireless transport system, support for multiple transport protocols is achieved by the nature of the transport system itself. ASK registers to be an application to the transport system to send and receive messages. It further registers one address for each of the protocols it is serving in order to interact with users on those protocols. For example, it can register ask@yourcomany.com for e-mail and 1234567 for SMS. Then ask@yourcompany.com and 1234567 become the URIs for their respective protocols similar to http://yourcompany.com is to the Web world.

ASK itself does not consider the incoming protocols; it is designed to send and receive messages by the means that it is registered to use. The payload (content) of the messages are what ASK interprets and acts upon.

10.5.3.2 The asynchronous nature of messaging protocols

ASK builds logic similar to an HTTP listener to present synchronous semantics over asynchronous protocols. It achieves this by acting as a client to the service the device requested. ASK makes a request to the service on behalf of the user, waits and processes the response from the service, then formats the response and presents it back to the users. Users have the illusion of a response from an earlier request.

10.5.3.3 Supporting Sessions

Upon receiving requests from a user, ASK create a session for the user. This allows conversational applications to function. Unlike in HTTP where session info is kept by the browser (or cookie), all session states are kept in the backend by ASK.

10.5.3.4 User Navigation

ASK transforms elements such as forms or menus, and presents a navigation command for end users. When elements such as forms are returned by a service, ASK retains the format of the form in the backend, and determines the action to take when the form is submitted with all other necessary information. When this user (using the set of command specified by ASK) fills and submits the command, ASK makes a request (based on the current user information stored in ASK) and processes the result again on behalf of the user. You can think of it as if the hyperlinks are stored in the backend when a user clicks on the "face" of the link.

10.5.3.5 Naming/Addressing an Application

Just as assigning a URL to a service in the Web world, to use ASK, a short name must be assigned to services to be ASK enabled. For example, assume the stock quote service has been assigned the path: /finance/quote and can be accessed as http://mycompany.com/finance/stock. Through Content Manager, a short name can be assigned to it (for example, *st*.). Now any messages ASK receives that begins with *st* signals a request for the stock quote service. A user can send *st orcl* to a site-wide address to which ASK is configured to listen, ask@mycompany.com for e-mail or 1234567 for SMS, and get back the stock quote for Oracle Corporation.

It is also possible to specify service-level address to services. Through Content Manager, one can also associate (e-mail) stock@mycompany.com and (SMS) 123FINANCE to the stock quote service. Once this is done, sending just *orcl* as an email to stock@mycompany.com or as SMS to 123FINANCE would result in receiving the stock quote of Oracle Corporation.

10.5.3.6 ASK Request Authorization

ASK differentiates the user of a request into two categories: guest or registered. Upon receipt of a user request by ASK, the source address of the request message is used to reverse-lookup a Oracle9*i*AS Wireless user for authentication. A user object can be located if a user has a device address, registered under his profile, the same as the source address of the request message. The located user object is then bound to a newly authenticated session created by the request. Otherwise, a guest user object is bound to the session. Whichever services are authorized to the user will be accessible to requests issued from the device.

Only those services belonging to the guest group are accessible to a guest user. Accessing a non-guest service triggers a returned form challenging the user for name and password. A valid Oracle9*i*AS Wireless username / password supplied by the user enables the previous session to be upgraded to an authenticated one with the user object identified by the name to be bound. Alternately, a guest user can log in explicitly through a login command, '!L', to avoid ever being challenged.

10.5.3.7 User interface and navigation commands

As discussed earlier, messaging clients typically only present plain text and do not offer conversational navigation capabilities. Recall that ASK transforms and formats responses from services to a certain presentation to enable such capabilities. ASK includes a set of presentation formats and navigational commands similar to what a Web browser has done for the Web world. Hence when a user invokes services using ASK, he or she would see the response in the format transformed by ASK. Further interactions with ASK would have to comply with the format expected by ASK. In this section we will be discussing commands users can issue to ASK. To issue a command is simply sending a message with the correct format. The command text can be put into a subject line or message body.

10.5.3.7.1 System Commands

- !H: (Help command) provides general help on the command usage
- !E: (Escape command) clears current form state.
- !S: marks the end of command sequence. A message may contain a sequence of commands, each separated by a line feed or command delimiter. !S marks the end of a command sequence. No interpretation will be done on text past the !S mark.
- help: the service level help. If no parameter is provided, all the async service help is provided. User can also provide a service short name as the parameters to acquired the help on a particular async service.

- !L <username> <password>- to sign on to the system with the user name and password.
- !O - to terminate a session

10.5.3.7.2 Service invocation commands These are commands to do service invocation, menu selection and parameters filling. There are no reserved command symbols for the service invocation and form commands. Certain commands, such as form command and menu item selection, can be invoked only when there is a current form / menu state maintained in the user's session. More details on form / menu state will be discussed later in this chapter.

- [<shortname> | <menuitem] <parm1><parm2> . . . to invoke a service. The first field provided could be a service short name or a menu item number. A menuitem can be provided only when the user previously received a menu message from a service result. The menu state is maintained in the user session of ASK. A user can make a selection based on the menu to trigger further actions. More detail on current menu state is explained later in this chapter.
- <parm1><parm2> . . . to fill the parameter of a form. When a user invokes a service without providing a required parameter, a form may be returned requesting the user to fill in the parameter values. This creates a current form state in the user session, expecting the user to send the parameter sequences in the subsequent command. The parameter values should be supplied on the command line in the same sequence as the parameters listed on the previously returned form.

10.5.4 Examples on Service Invocation

10.5.4.1 Invoke by Service Short Name

All services that are ASK-enabled should be assigned short names to be accessed by the end user. The short name should be able to uniquely identify a service on the entire site. To invoke a service, a message should be sent to a site-wide address, such as *info@oraclemobile.com*, to which the ASK is configured to listen. The command line has the format:

```
<Svc Short Name> <parm1> <parm2> . . .
```

In the following example, a message is sent to the site-wide address: *info@oraclemobile.com*, to invoke a stock quote service whose short name is *ST*. The service requires a stock symbol as its parameter (in this case, *ORCL* is provided).

Figure 10–12 Invoking by service short name

10.5.4.2 Invoke by service associated device address

Each service may have some device address associated with it. For example, an e-mail address *stock@oraclemobile.com* can be used to identified a stock service. Since the service has been identified in the destination address of the request message, there is no need to specify the service short name in the command line. Only the service parameters are required in the command line, for example, the stock symbol.

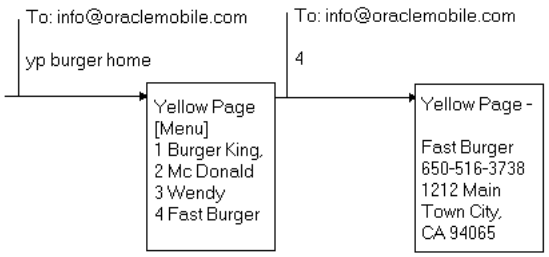
All of the system commands (for example, 'help') can still be issued to the service-associated address. They are interpreted by Async Server in the same way they are sent to the site-wide address.

Figure 10–13 Invoking by service associated device address

10.5.4.3 Menu Capability

The way the features are presented is similar to the HTTP model. A service invocation may trigger the return of a message with the menu. Each menu item is prefixed with a number. Users are able to make selections by issuing another message in which the message content contains the menu item number. This extends the service capability for much better user interaction. A yellow pages service having a short name of *yp* expects two user parameters, *category* and *area*. Users invoke services by providing the values, for example, *burger* and *home* (a landmark for the user). The application searches for all the Burger stores in the home area. A returned message from the service result contains a name list of Burger stores. The user then issues another message to get detailed information about the stores in which he is interested.

Figure 10–14 Menu capability



10.5.4.4 Form Capability

A form is the result of a service invocation requesting user input. The ideal user interaction for ASK is to have the user fill in the input parameters on the command line instead of having to fill in the form, which requires more message round trips.

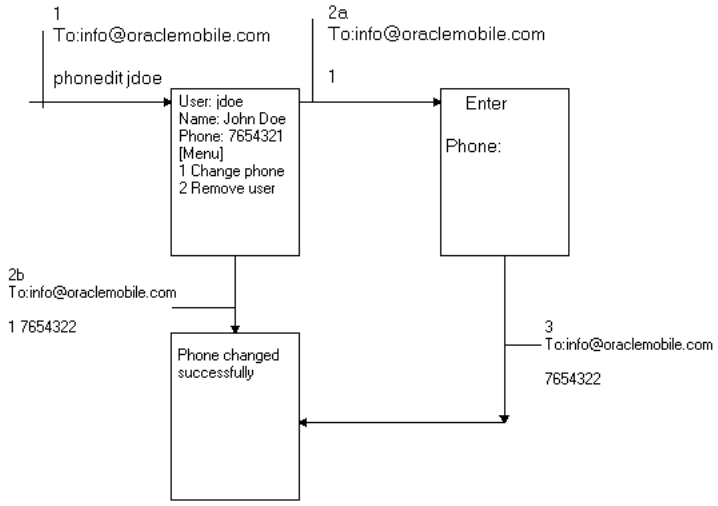
Figure 10–15, "Form capability" demonstrates the possible interaction of a *phone book* service. The *phonedit* command enables users to search and edit the phone number for a particular user. It expects a name as its parameter. *jdoe* is provided in the example. The information of *jdoe* is returned with a menu, enabling the device user to edit the phone number or remove the user. There are two options for editing the phone number:

1. Make a selection without entering any parameter: This is represented in box 2a. A form is returned prompting the user to enter the new phone number. The device user creates a new message with the message body containing the new phone number.

Or

2. Enter the selection with the required parameters. Box 2b demonstrates the scenario. The device user is aware that a form should returned in response to their selection 1 (Change phone). Therefore, the parameter value (phone number) is supplied together with the selection. This saves a message round trip.

Figure 10–15 Form capability

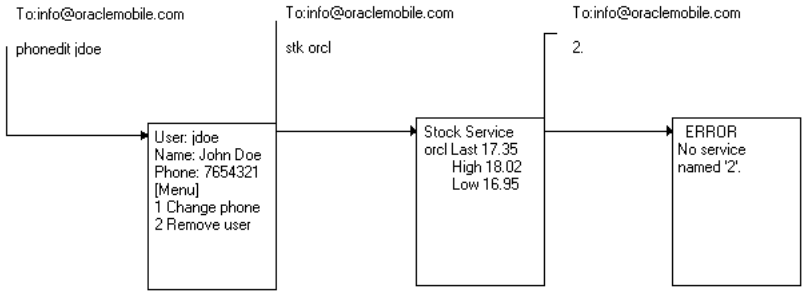


10.5.4.5 Current Menu State

Since a session is maintained for each user, menu navigation is made possible. The term *current menu* identifies the latest menu a user received from the ASK. The state of the current menu is kept in the user session on ASK. Whenever a menu selection is made by a user, it applies to the current menu. If a menu has not yet been received for the user, the ASK will attempt to locate a service whose short name is the same as the number provided by the user. An error is returned when no such service is found.

A service invocation through short name or device address automatically cancels the menu state created by the previous service invocation. The figure below demonstrates the situation. A menu returns as a response to invoking the *phonedit* service. A message for requesting the *stk* service is subsequently issued. It clears the menu state created by the invocation of the *phonedit* service. An attempt to make a menu selection triggers an error message from the ASK.

Figure 10–16 Current Menu State

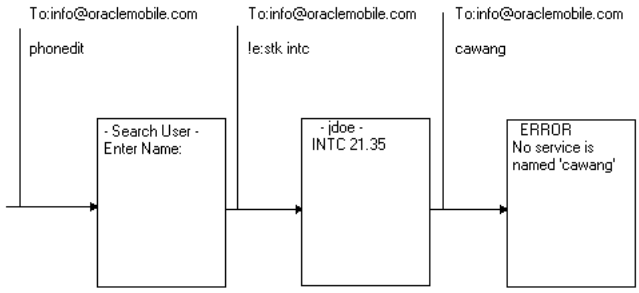


10.5.4.6 Current Form State

A current form state is created in the user session whenever the user receives a form message. Subsequent form parameter values can be issued by the user to fill the parameter requested from the previous form message. If the user decides not to fill the form but to invoke another service, the *Escape* command can be issued to cancel the current form state. Once the form state is clear, any form parameters issued by the user are considered invalid. An error message should be returned in respond to a form parameter without a current form state.

Figure 10–17, "Current Form State" illustrates a form state example. The device user invokes the *phonedit* service without providing any parameters. A form message is returned to the user expecting the user to fill in the search name. If the device user changes his / her mind and decides to invoke another service (for example *stk*) the first step is to clear the form state so that ASK will not treat the command *stk* as the name value expected from the phonedit service. Then, a new *stk* command can be issued. These two steps are combined into one message by separating the two commands with the default command separator (;).

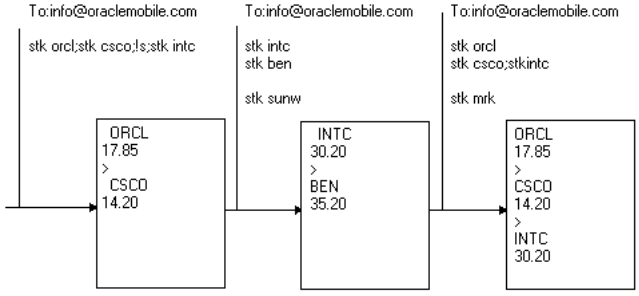
Figure 10–17 Current Form State



10.5.4.7 Multiple commands in one message

Multiple commands can be issued from one message. They can be issued from the same line, each command separated by the configurable command separator (default [;]). Or, commands can be on different lines. The first blank line or *stop* command (*!s*) encountered, marks the end of the command sequence. No command interpretation will be done on text after the mark.

Figure 10–18 Multiple Commands in One Message



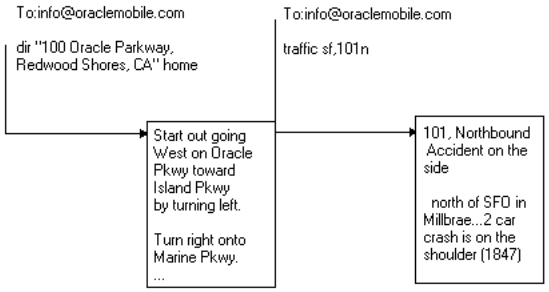
10.5.4.8 Parameter separator

Multiple parameters may be required for an Async service.The default parameter separator is a blank space. If a parameter value contains space within it, it can be enclosed by double quote to represent a single parameter value. The parameter separator is configurable at the service level.

The example below illustrates a direction service expecting both the *from* and destination (the *to*) addresses. The *from* address is provided with double quotes to enclose the whole value. The destination is supplied as a landmark, *home* from the

user profile. The second message sent from the user is to request traffic information service. The service is configured to use a comma (,) as the parameter delimiter; users provide the parameter values with (,) to separate them.

Figure 10–19 Parameter Separator



10.5.5 Writing Asynchronous Applications

The way to develop applications for ASK is basically the same as for the Device Portal. Service Provider receives user parameter from the device, and responds with the result (in Mobile XML format). The requirement on the ASK client is low; the ability to send and receive text messages. Therefore, only a subset of the Mobile XML tags are applicable to ASK, as shown in Table 10–6, "Summary of semantics for MobileXML tags".

Developers may choose to have a different logic flow (for example, rendering the results differently) for the ASK device. In this case, they would need to be able to recognize if the request was coming from an ASK device class. This is accomplished by checking the device class attribute of the user request. The request from ASK has the device class attribute value of either *messenger*, or *micromessenger*. The information can be acquired from the input arguments for a service written in adapter form, or the HTTP header for services based on HTTP / OC4J adapter. The input argument *_DeviceCategory* defined in the ServiceContext specifies the device class value for adapter formed services. For HTTP / OC4J based services, the value can be picked up through the HTTP header *x-oracle-device.class*.

Similarly, any section of the ASK specific Mobile XML result, created by the application, binds the value of *messenger* or *micromessenger* to the element attribute *deviceclass*. ASK processes elements common to all devices (with no value specified in *deviceclass*), or elements with the attributes containing the value of *messenger* or *micromessenger*.

10.5.5.1 ASK enabling MobileXML Application

All mobile XML service can be made ASK enabled from a technical standpoint. The user experience while using ASK is worth considering when deciding how to build an application or ASK enabling an existing application. This is the same practice you might have been applying to decide how you want to render you application to different types of devices (screen size, form factor and such). ASK assumes a client with plain text input so it is even more appropriate to consider user experience. Services that expect many user interactions or have a complicated UI may not work well.

In addition, some of the Mobile XML tags do not make sense for ASK and one should be aware of the specific semantics ASK has for the set of XML tags. Since ASK do not assume any sort of client side browsing capability, it is common that tags which assumes certain keys or actions on the device are not appropriate for ASK. The following table lists all the tags for MobileXML, as well as their semantics in the context of ASK.

Table 10–6 Summary of semantics for MobileXML tags

Mobile XML Tag	Semantics
SimpleAction (MobileXML)	Treated the same as the SimpleMenuItem and SimpleHref. Each SimpleMenuItem, SimpleHref or SimpleAction will be prefixed with a number in the device result for async user to make selection.
SimpleAudio (MobileXML)	Ignored—not applicable to async devices.
SimpleBind (MobileXML)	Ignored—not applicable to async devices.
SimpleBreak	A new line is created on the page.
SimpleCache (MobileXML)	Ignored—not applicable to async devices.
SimpleCase (MobileXML)	Ignored —not applicable to async devices.
SimpleCatch (MobileXML)	Ignored— not applicable to async devices.
SimpleCol	Output the text.
SimpleContainer	Processed—no output is generated.
SimpleDefault (MobileXML)	Ignored—not applicable to async devices.
SimpleDisconnect (MobileXML)	Ignored—not applicable to async devices.
SimpleDisplay (MobileXML)	Ignored—not applicable to async devices.

Table 10–6 Summary of semantics for MobileXML tags

Mobile XML Tag	Semantics
SimpleDTMF (MobileXML)	Ignored—not applicable to async devices.
SimpleEM (MobileXML)	Output the text.
SimpleEvent (MobileXML)	Ignored—not applicable to async devices.
SimpleExit (MobileXML)	Ignored—not applicable to async devices.
SimpleFinish (MobileXML)	Ignore—not applicable to async devices.
SimpleFooter (MobileXML)	Ignored.
SimpleForm	The form state is maintained in the server so the parameters issued by the user can be paired with their corresponding keys.
SimpleFormItem	The item text is printed on the returned page. User fills the corresponding item values in the same sequence as the item presented on the page.
SimpleFormOption	A list of form options is printed on the returned page with a number prefixed each form option. The user can fill the select item by giving either the prefix number or the option text. For example, a select item of 'State' should contain the option, '1 AL, 2 CA, 3 UT...'. The user can supply the value of '2' or 'CA' to select the option 'CA'. Only Radio box (single selection) is supported on the version.
SimpleFormSelect	Output the text.
SimpleGo (MobileXML)	Ignored—not applicable to async devices.
SimpleGrammar (MobileXML)	Ignored
SimpleHeader (MobileXML)	Ignored.
SimpleHelp	Output the text.
SimpleHref	This is treated the same as SimpleMenuItem. All the SimpleMenuItem is prefixed with a number so the user is able to select the item by responding with the corresponding number.
SimpleImage	Ignored; not applicable to ASK.
SimpleKey (MobileXML)	Ignored—not applicable to async devices.
SimpleMatch(MobileXML)	Ignored—not applicable to async devices.

Table 10–6 Summary of semantics for MobileXML tags

Mobile XML Tag	Semantics
SimpleMenu	A new line is created on the page. The menu state is maintained in the server.
SimpleMenuItem	The value of the menu item is printed on the returned page with a number prefix to identify the menu item. The target url and the number prefix is stored in the server so the url can be retrieved after the user makes the selection.
SimpleMenuItemField (MobileXML)	Output the text.
SimpleMeta	Ignored—not applicable to async devices.
SimpleMItem(MobileXML)	Ignored—not applicable to async devices.
SimpleName (MobileXML)	Ignored—not applicable to async devices.
SimpleOptGroup (MobileXML)	Ignored—not applicable to async devices.
SimplePrev (MobileXML)	Ignored—not applicable to async devices.
SimpleProperty (MobileXML)	Ignored.
SimpleRefresh (MobileXML)	Ignored—not applicable to async devices.
SimpleReprompt (MobileXML)	Ignored—not applicable to async devices.
SimpleResult	Processed—no output is generated
SimpleRow	Print a new line to the returned page.
SimpleSpeech (MobileXML)	Ignored—not applicable to async devices.
SimpleStrong (MobileXML)	Output the text
SimpleTable	No op
SimpleTableBody	Output new Line
SimpleTableHeader	Output a new line.
SimpleTask (MobileXML)	Ignored—not applicable to async devices.
SimpleText	Print a new line on the returned page.
SimpleTextItem	Output the text.
SimpleTimer	Ignored—not applicable to async devices.

Table 10–6 Summary of semantics for MobileXML tags

Mobile XML Tag	Semantics
SimpleTitle (MobileXML)	Output the text.
SimpleValue (MobileXML)	Ignored—not applicable to async devices.

10.6 Runtime and Data Model APIs

This section is for advanced users.

10.6.1 Oracle9*i*AS Wireless Runtime

Oracle9*i*AS Wireless runtime layer is a servlet in the OC4J servlet container. Oracle9*i*AS Wireless runtime processes requests from Hypertext Transaction Protocol (HTTP) user agents, async user agents (such as SMS, e-mail, two-way pagers), and autonomous mobile 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 Oracle9*i*AS Wireless.

10.6.1.1 Session Management

The Oracle9*i*AS Wireless runtime tracks the runtime session independently of the Servlet session, by rewriting every URLs with an added parameter "PAsid," which specifies the session id. The session tracking identifies that a sequence of requests are submitted by the same device. The runtime session contains the user information, authentication contexts, adapter contexts, runtime contexts, URL caches, and other states essential for the context sensitive services.

WAP 2.0 devices that implement the WAP HTTP State Management Specification (http://www.wapforum.org/) can support cookies for session management. Most of the commercial WAP gateways manage persistent cookies on behalf of the devices. If the device or gateway does not support cookies, the OC4J servlet container falls back to URL rewriting for session tracking. Since the Oracle9*i*AS Wireless runtime also tracks the session, it is possible for more than one runtime session to be bound to a single servlet session. For example, two Netscape browsers on the same client PC can open two independent runtime sessions although the browsers share the same servlet session because of the shared cookie repository.

By default, the binding to the Servlet session is enabled and is necessary for the OC4J load balancing and fail over facility. The runtime session states are replicated to other OC4J instances in the "island" so that the device requests can be redirected

to another OC4J instance in the island when the first instance fails. The runtime sessions that are bound to the servlet sessions are invalidated when the servlet sessions expire.

The session binding from the runtime session to Servlet session can be disabled by the parameter setting “enable.http.session.binding=false” in the System.properties file. Without binding to the servlet sessions, the runtime sessions are expired when the sessions remain idle for more than what is specified by the “wireless.session.expiration.time” parameter in the site configuration.

Every request from the device is serviced within the context of a valid runtime session. The requests from anonymous devices are also tracked and assigned to individual runtime sessions although the owners of the sessions are the same Guest user, which is an anonymous user. The Oracle9*i*AS Wireless runtime automatically provisions a “virtual” user in the Wireless repository for each device that can be consistently identified, using the identifiers available in the devices. Runtime sessions for virtual users are opened whenever the device identifiers are present in the requests. The device identifiers may be based on native device identifiers such as the Mobile Identification Number (MIN), Mobile Subscriber ISDN (MSISDN), Ipv6 Address, Electronic Serial Number (ESN), etc. The device identifiers may be also provisioned into the device by the WAP gateway. The WAP Client ID Specification (http://www.wapforum.org/) defines a standard scheme for supporting the device identifiers. If no device identifiers are supplied in the request, the Oracle9*i*AS Wireless runtime provisions the device identifiers into the devices using the persistent cookies whenever possible.

The Oracle9*i*AS Wireless runtime uses the device identifiers only to facilitate personalization under the virtual user. The runtime sessions opened under the virtual users have access to the information such as personalized presets and customization profiles in the repository. The device identifier also enables the device to reconnect to the same runtime session for the user, as long as the session has not expired. The device identifiers add robustness to the session management for Oracle9*i*AS Wireless, enabling continuity of the service in the face of intermittent connection losses. The users may also make telephone calls in between connections to the Oracle9*i*AS Wireless without losing their contexts.

Device identifiers are not a mean for authentication. Although the runtime sessions for the virtual users are not authenticated, it does not prevent the users from accessing their personalized portals. The users may establish authenticated sessions only if they register with the Oracle9*i*AS Wireless. The user can supply the user name and password during the registration. The user’s personalization profiles and presets are still available to the user after the user becomes registered. The

advantages of the registration include the authentication process that gives access to the secured services, such as the e-Wallets and financial transaction services.

The application programs for the services that require the authenticated sessions must add the “PAlogin=true” parameter in the URLs. When the Oracle9*i*AS Wireless runtime detects the PAlogin=true parameter among the URL parameters in the request for a service, the runtime tries to authenticate the user if the runtime session is not already authenticated. The authentication process, which typically involves the user supplying the user name and password to the Oracle9*i*AS Wireless Single Sign On (SSO) Server, is performed before the runtime invokes the service being requested. See Section 10.6.2.15, "User-Defined Hooks Examples" for how the folder renderer service can be used to prepare the URLs with “PAlogin” parameter for the secured services in a folder. After the “PAlogin” parameter invokes the authentication process, the application programs for secured services still have the responsibility to check that the session is authenticated. The applications that has direct access to the Oracle9*i*AS Wireless runtime objects can use isUserAuthenticated() method in oracle.panama.rt.Session interface. Applications written for the HttpAdapters can get the information from the Http header attribute "x-oracle-user.authkind" which has the values "authenticated" or "unauthenticated."

In addition, the applications can also check if the session is secured by the SSL, TLS, or WTLS channels. The application that has direct access to the Oracle9*i*AS Wireless runtime objects can use isSecure() method in the oracle.panama.rt.Request interface. Applications written for the HttpAdapters can get the isSecure() condition through the HTTP header attribute "x-oracle-device.secure," which has the values "true" or "false."

The authorization for access to a service is performed for each request for all authenticated or unauthenticated sessions. The authorization makes sure that the session user has the privilege to access the service. The default authorization policy does not differentiate whether the session is authenticated or unauthenticated. The unauthenticated sessions of a “virtual” or “registered” user has as much visibility as the authenticated sessions. It is therefore critical for the applications to apply the “PAlogin” parameter to enforce the authentication.

10.6.1.2 Virtual User Concept

The Oracle9*i*AS Wireless runtime automatically provisions “virtual” users in the Wireless repository for the devices that can be consistently identified, using the identifiers available in the devices. The virtual user option gives the device owners immediate access to the personalization features of the portal, which enhance the

user experience. It automates the provisioning process for the carrier and enterprise portal administrators using the emerging WAP Client ID standards.

The device owners can register with Oracle9*i*AS Wireless to gain access to secured services through authentication. The registration can be done from the setup menus by the device owner. This self-provisioning registration feature further simplifies the administration tasks. The devices with the virtual user support let the registered users connect to Oracle9*i*AS Wireless and access the personalized services without signing on to the system until they are requested by the secured services to authenticate. The virtual user feature not only improves the accessibility of the portal but also enhances the data mining capability of portal operators since the activities of the devices can be identified with virtual identities.

The virtual user feature can be disabled by the site wide configuration parameter setting “wireless.virtualuser.enabled=false.” This property can be modified by the **Enable Virtual User** option in **System Manager>Site>User Provisioning** control panel. If the virtual user feature is disabled or if the device does not support device identifier, then the session is opened under the “Guest” user, which must be provisioned in the repository. The Oracle9*i*AS Wireless bootstrap repository includes the anonymous user “Guest.”

Applications that have direct access to the Oracle9*i*AS runtime objects can check the value of oracle.panama.model.UserType returned by the getUserType() method in oracle.panama.model.User. The User of the runtime session can be retrieved from the getUser() method in oracle.panama.rt.Session. Applications that are written for the HttpAdapter can get the user type information from the HTTP header attribute "x-oracle-user.userkind." The possible values of this attribute are "anonymous," "virtual,” or "registered."

10.6.1.3 Runtime API

During the request execution, the Oracle9*i*AS Wireless runtime dispatches the authentication, authorization, device identification, location acquisition, data logging and other business logics to the respective plug-in modules.

Oracle9*i*AS Wireless Runtime API provides the Java interfaces to examine the runtime execution states, monitor the runtime execution behavior, and augment the default execution semantics. The Runtime API consists of four Java packages:

- oracle.panama.rt provides the interfaces to the essential runtime objects for state examination.
- oracle.panama.rt.event provides the interfaces to monitor the runtime execution sequence based on the Java event model.

- oracle.panama.rt.hook provides the interfaces for the essential runtime customizable components and the default implementation policies for these interfaces.
- oracle.panama.rt.xform provides the interface for the customizable transformers for plug in as the Device's transformers.

These four packages are included in the wireless.jar file. Make sure you have included wireless.jar in your Java classpath when you compile your Java application or plug-in modules that depend on the Runtime API.

10.6.1.4 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 custom listener registration module may subscribe the listeners only to the requests for the billable services. Such a listener may add business rules to the runtime controllers.

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

- Examining and modifying the state of the runtime objects during the runtime execution (`oracle.panama.rt`). The essential runtime objects are:
  - Request: the service invocation specification
  - Response: the result of a request execution
  - Session: the durable context of a connection
  - ServiceContext: the context in which the request is being executed
  - ManagedContext: the application context of a particular service
- Monitoring the runtime execution sequence, based on the event model (oracle.panama.rt.event)
  - RequestListener observes the Request Events
  - ResponseListener observes the Response Events
  - SessionListener observes the Session Events
- Extending the customizable behavior of the runtime controllers using hooks and policies (oracle.panama.rt.hook)

- Hooks are the main extension mechanisms.
- The default implementations of hooks are provided among the public API as policies.
- Policies can be delegated to by the customized implementation of hooks, to realize the chain of responsibility design patterns.
- The customized hooks must implement the static getInstance() method according to the Oracle9*i*AS Wireless singleton pattern.

The new customized hook implementations, which implement the respective interfaces and the singleton pattern, are registered in the appropriate entries through **System Manager>Site>Wireless Web Server>Hooks** control panel in the Webtool.

10.6.1.5 Runtime Objects

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 classes in this package are:

- RequestFactory
- SessionHolder

10.6.1.6 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, Oracle9*i*AS Wireless runtime parses the URL query string to retrieve the parameters. The runtime agents or other internal clients can set these parameters programmatically. Since Oracle9*i*AS Wireless runtime may cache and rewrite the URL for HTTP user agents, some of the parameters are maintained in the URL cache for the user. Oracle9*i*AS Wireless runtime may have to parse both the query strings from the HTTP request and the URL cache to build a complete list of query parameters.

Step 2 in Section 10.6.2.1, "Case: A Request Involving Session Establishment and Authentication" shows that each time a new request object is created, Oracle9*i*AS Wireless runtime passes the request object to the ListenerRegistrationHook to let the hook register listeners.

The following table 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 Mobile XML results can contain the runtime variables, (composed from the names of the parameters) by appending two underscore characters (__) before and after the parameter name. These runtime variables in the Mobile XML results are "place holders" which are replaced by the values of the parameters during the post processing phase (Step 25 in Section 10.6.2.1, "Case: A Request Involving Session

Establishment and Authentication") before the final result is returned to the requester.

Table 10–7 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"	Deprecated
Request.PASSWORD	"PApassword"	Deprecated
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.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 authentication request.
Request.SESSION_HOME	“PAhome”	The object id of the service to be set as the session home.
Request.GO_SESSION_HOME	"PAgoHome"	A request parameter to invoke the session home service.
Request.REQUEST_USER_PROFILE	"PArprof"	The parameter used to specify the user profile for the request.
Request.SESSION_USER_PROFILE	"PAsprof"	The parameter used to specify the user profile for the session.

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:

```
public void invoke(ServiceContext sc) {
    .
    Request request = sc.getRequest();
    String value = request.getParameter(Request.REQUEST_LANDMARK); [4]
    request.setParameter(Request.SESSION_LANDMARK, “Redwood City”); [5]
    .
```

```
}
```

10.6.1.7 Response

This interface represents the Response objects in Oracle9*i*AS Wireless runtime. A listener can subscribe to events from a Response. The Response object is the execution result of the prior Request object.

10.6.1.8 Session

This interface represents the session objects in Oracle9iAS Wireless runtime. A valid session is established after an anonymous user, virtual user, or registered user is identified for the session (refer to the Session Management Section above for the user identification process). Any request (or service invocation) can only be executed in a valid session context. A session can either expire after the session exceeds the maximum 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.

Step 7 in Section 10.6.2.1, "Case: A Request Involving Session Establishment and Authentication" shows that each time a new session object is created, Oracle9*i*AS Wireless runtime passes the session object to the ListenerRegistrationHook to let the hook register listeners.

10.6.1.9 ServiceContext

A ServiceContext provides the service request context for a valid and authorized request. A new ServiceContext object is created for each validated request. The ServiceContext stores the input parameters, output parameters, and Mobile XML results. The associated request and session can be accessed from the ServiceContext object.

The Mobile XML result can contain the system defined ServiceContext parameters using the runtime variables as "place holders," which are substituted with values during the post processing phase (Step 25 in Section 10.6.2.1, "Case: A Request Involving Session Establishment and Authentication") before the final result is returned to the requester.

Runtime variables are composed from the names of the parameters, by appending two underscores (__) before and after the parameter name.

Example:

Mobile XML results can contain runtime variables as follows:

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

Given the variables above and if the following three conditions exist in the ServiceContext:

- 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, and Mobile XML result in the ServiceContext are externalized as an XML document.

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

The following table 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 10–8 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/r m?PAoid=100,	then the URI of the servlet is:	/ptg/rm.
ServiceContext.HTTP_REQUEST_NAME	"_HTTP_REQUEST_ NAME"	The absolute URL of the portal servlet requested through the HTTP protocol.
ServiceContext.HTTPS_REQUEST_ NAME	"_HTTPS_REQUEST_ NAME"	The absolute URL of the portal servlet requested through the HTTPS protocol.

Table 10–8 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.ABS_REQUEST_NAME	"_ABS_REQUEST_NAME"	The page-name of the portal servlet requested, for example: http://www.oracle.com/ptg/rm
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.
ServiceContext.COUNTRY	"_Country"	The country that contains the current location.
ServiceContext.STATE	"_State"	The state that contains the current location.

Table 10–8 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.POSTALCODE	"_Postalcode"	The postal area that contains the current location.
ServiceContext.DEVICE_CATEGORY	"_DeviceCategory"	The category of the device, possible values are
"voice"	"microbrowser"	"pdabrowser"
"pcbrowser"	"micromessenger"	"messenger"
ServiceContext.SOFT_KEYS	"_SoftKeys"	The softkeys supported by the device.
ServiceContext.IMAGE_PREFERENCES	"_ImagePreferences"	The image preferences of the device.
ServiceContext.MAX_DOC_SIZE	"_MaxDocSize"	The maximum size of the document handled by the device.

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 the above table. Line **[7]** shows that the parameters from the request object are also available among the ServiceContext arguments. However, the ServiceContext parameters are not part of the PARAMETERS attribute category in Request objects, and are not accessible from the Request objects. They can be accessed only from the ServiceContext arguments as shown in the following example.

Example:

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

The Java program constants 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 Mobile XML result. The “Arguments” tag is a sibling of the “Result” tag; it contains all input and output arguments.

Table 10–9 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 the Mobile XML result.

The following example of the XML document for a ServiceContext shows the “ServiceRequest” tag as the root element of the ServiceContext. 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>

   <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>
               </SimpleResult>
         d. </Result>

2.</ServiceRequest>
```

10.6.1.10 ManagedContext

In many situations, the customized hooks, listeners, and adapters require session-long, application-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 application-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 Oracle9*i*AS Wireless runtime when the session terminates.

10.6.1.11 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.

10.6.1.12 SessionHolder

The SessionHolder class is defined in the oracle.panama.rt. package. The SessionHolder is the serializable representation of the runtime Session. It is used to bind the runtime Session to the servlet session as required for the OC4J cluster configuration. Only serializable objects placed in the runtime session and the servlet session are replicated among other OC4J instances in the island. The portal

developers can get an instance of this serializable object using the getSessionHolder() method in the Session.

10.6.1.13 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:

```
 public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {
      Request req = RequestFactory.createRequest(request, response); [3]
      if (req == null) {
            return;
      }
      try {
            Response resp = req.execute(); [8]
      } catch (Exception ex) {
     } finally {
            req.invalidate(); [11]
     }
}
```

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 from the servlet.

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

10.6.1.14 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 through the Oracle9*i*AS Wireless runtime.

```
import oracle.panama.rt.RequestFactory;
import oracle.panama.rt.Request;
import oracle.panama.rt.Response;
import oracle.panama.rt.Session;
import oracle.panama.rt.ServiceContext;
.
import oracle.panama.model.MetaLocator;
import oracle.panama.model.ModelServices;
import oracle.panama.model.Service;
import oracle.panama.model.User;
import oracle.panama.model.AlertAddress;
.
.
.

Session signon(String user, String password) throws PanamaException {
     Request request = RequestFactory.createRequest(user, password);
     request.validate();[17]
     Session session = request.getSession();[18]
     request.invalidate();
     return session;
}

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

     try {
```

```
          req = RequestFactory.createRequest(session, service,
                                          user, address);[28]
          if (req == null) {
               return null;
          }
          try {
               req.setParameter("TickerSymbol", symbol);
               sc = req.validate();
               resp = req.execute();
           if (sc.isAnyResultPresent()) {
               content = resp.getContent();
           }
       } catch (Exception ex) {
       } finally {
           req.invalidate();
       }
           return content;
       }
}

String userName = "orcladmin";
String password = "manager";
String effectiveUserName = "Guest";
String symbols[] = { "orcl", "sunw", "csco" };

void main() {
     ModelServices models = MetaLocator.getInstance().getModelServices();
     User user = models.lookupUser(effectiveUserName);
     Service service = models.lookupService("YahooQuote");
     AlertAddress[] addresses = user.getAddresses();
     Session session = signon(userName, password);
     for (int i = 0; i < symbols.length(); i++) {
         .
         .
         String content = invokeService(session, service, user, addresses[0],
                              symbol[i]);
         .
         .
     }
}
```

The signon() method signs on the user to the Oracle9*i*AS Wireless 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 Oracle9*i*AS Wireless 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.

10.6.1.15 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 execution. 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 ListenerRegistrationHook subscribes the listeners to receive events from the subject, such as Request, Response, or Session.

10.6.1.16 Implementing the RequestListener Interface

The implementor of `oracle.panama.rt.event.RequestListener` can receive 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 10.6.2, "Reference Model". However, not all the Request-related events will be generated. Which specific Request-related event will be generated is controlled by the enent mask in System Manager -> Site -> Wireless Web Server -> Event and Listeners control panel in the Webtool.

For example, if you want to have your RequestListener receive the request begin event, you should set the Enable 'request begin' Event to true in the **System Manager -> Site -> Wireless Web Server -> Event and Listeners** control panel in the Webtool. The site configuration property names are:

```
wireless.http.event.beforeRequest
wireless.http.event.requestBegin
wireless.http.event.requestEnd
wireless.http.event.serviceBegin
wireless.http.event.serviceEnd
wireless.http.event.transformBegin
wireless.http.event.transformEnd
wireless.http.event.requestError
wireless.http.event.afterRequest
```

Step 11 in Section 10.6.2.1, "Case: A Request Involving Session Establishment and Authentication" indicates that the RequestListener can intercept the input parameters during the requestBegin(RequestEvent) and apply additional business rules to the request parameters before service invocation.

10.6.1.17 Implementing the ResponseListener Interface

The implementor of oracle.panama.rt.event.ResponseListener can receive the Response-related event. The only possible Response-related event is response error. If you want Oracle9*i*AS Wireless runtime to have your ResponseListener receive the

response error event, you should set the Enable 'response error' Event option to true in **System Manager -> Site -> Wireless Web Server -> Event and Listeners** control panel in the Webtool. The site configuration property name is: wireless.http.event.responseError

10.6.1.18 Implementing the SessionListener Interface

The implementor of oracle.panama.rt.event.SessionListener can receive the Session life cycle events. The possible Session events include:

- `before session`
- `session begin`
- `session authenticated`
- `session end`
- `after session`

The timing sequence regarding when the event is generated is discussed in Section 10.6.2, "Reference Model". However not all the Session events will be generated. Which specific Session event will be generated is controlled by the event masks in the System Manager -> Site -> Wireless Web Server -> Event and Listeners control panel in the Webtool.

For example, if you want to have your SessionListener receive the session begin event, you should set the *Enable 'session begin' Event* option to true in the **System Manager -> Site -> Wireless Web Server -> Event and Listeners** control panel in the Webtool. The site configuration property names are:

```
wireless.http.event.beforeSession
wireless.http.event.sessionBegin
wireless.http.event.sessionAuthenticated
wireless.http.event.sessionEnd
wireless.http.event.afterSession
```

10.6.1.19 Guidelines

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

1. Implement the RequestListener, ResponseListener, or SessionListener interface.
2. Compile the new Java source files from Step 1 with the wireless.jar file in the classpath.

3. Modify the event mask entries in the System Manager -> Site -> Wireless Web Server -> Event and Listeners control panel to enable the generation of specific events.
4. Specify the class names for the RequestListener, ResponseListener, and SessionListener in the System Manager -> Site -> Wireless Web Server -> Event and Listeners control panel of webtool. The site configuration property names are:

   ```
   wireless.http.locator.combined.listener.classes
   wireless.http.locator.session.listener.classes
   wireless.http.locator.response.listener.classes
   wireless.http.locator.request.listener.classes
   ```

5. Restart the Oracle9iAS Wireless instance.

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)`
- `sessionAuthenticated(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 10.6.1.16, "Implementing the RequestListener Interface". 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.

10.6.1.20 Hooks

The Oracle9*i*AS Wireless runtime specifies the hook interfaces for standard plug-in modules. The following sections describe the hooks in the order in which they are invoked by the runtime.

In the Oracle9iAS Wireless Runtime API, Hook and Policy form a chain of responsibility design pattern, within which Policy is the default implementation of Hook that can be delegated by the custom implementation.

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

Table 10–10 Classes that Implement the Default Policies

Hook Name	Policy Name
AuthenticationHook	AuthenticationPolicy
AuthorizationHook	AuthorizationPolicy
CallerLocationHook	CallerLocationPolicy
DeviceIdentificationHook	DeviceIdentificationPolicy
FolderRendererHook	FolderRendererPolicy
HomeFolderSorterHook	HomeFolderSorterPolicy
ListenerRegistrationHook	ListenerRegistrationPolicy
LocationServiceVisibilityHook	LocationServiceVisibilityPolicy
MobileIdHook	MobileIdPolicy
NormalizeAddressHook	NormalizeAddressPolicy
PostProcessorHook	
PreProcessorHook	
ServiceVisibilityHook	ServiceVisibilityPolicy
SessionIdHook	SessionIdPolicy
SignOffHook	SignOffPolicy
SignOnPagesHook	SignOnPagesPolicy

10.6.1.21 The ListenerRegistrationHook

Steps 2 and 7 in Section 10.6.2.1, "Case: A Request Involving Session Establishment and Authentication" show 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 billable services. Such a listener may add business rules to the runtime controller.

10.6.1.22 The SessionIDHook

The Oracle9*i*AS Wireless 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.

10.6.1.23 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 request.

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.

10.6.1.24 AuthenticationHook

The Oracle9iAS Wireless 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 in the Oracle9iAS Wireless Single Sign On (SSO) server.

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 Oracle9iAS Wireless 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.

10.6.1.25 SignOnPagesHook

In the Oracle9*i*AS Wireless environment, the sign on pages are generated by the SSO server. The SignOnPagesHook is used primarily for authentication against the stand alone repository. This hook is not shown in the execution because it is invoked only when the AuthenticationPolicy raises the AuthenticationFailOverException.

When the SignOnPagesHook generates the sign-on page, the Oracle9iAS Wireless runtime sends that sign-on page to the user, who submits the user's name and password for authentication by the default authentication module in stand alone mode (without SSO).

10.6.1.26 MobileIDHook

Runtime invokes the MobileIDHook to determine the mobile client ID.

The mobile identifier may be supplied by the external accounting system, by one of the fields (such as, the mobile ID field or external ID field) of the user object in the repository, or by one of the attributes in the HTTP header. The HTTP header attribute names can be specified by the "wireless.mobile.id.request.parameter.name" through the System Configuration Webtool. This mobile id is placed in the authenticated session.

10.6.1.27 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 Oracle9iAS Wireless 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.

10.6.1.28 CallerLocationHook

The CallerLocationHook provides the interface to acquire a caller's physical location in terms of latitude and longitude. The Oracle9iAS Wireless provides two different default implementations of the CallerLocationHook interface.

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

The oracle.panama.rt.common.LocAcq provides the automatic location acquisition implementation if the user specifies the appropriate privacy directive. If the automatic acquisition fails or is disabled through the *Enable Mobile Positioning* flag in the System Manager -> Site -> Location Management control panel of the

webtool or by setting the "wireless.elocation.mp.enable" parameter in the Site Configuration parameter table, the prior location mark 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.

10.6.1.29 Service

Services are Oracle9iAS Wireless 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 two hooks control how the content of a folder gets rendered:

- FolderRendererHook
- LocationServiceVisibilityHook

The FolderRendererHook uses the 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, accessible through the FolderRendererPolicy, combines the contents of the home folder with the folders and services from one or more of the user's groups. The LocationServiceVisibilityHook selects from the location-based subfolders in the folder for those whose regions intersect with the current position of the mobile device.

Each folder can be associated with a folder rendering service which provides customized view of the folder. The default folder rendering policy or the site wide customized folder rendering hook is used only if the folder does not have an associated rendering service, either assigned to it or inheritable from its parent folders.

10.6.1.30 PreProcessorHook, Transformer, and PostPorcessorHook

If the PreProcessorHook is specified, the Runtime invokes the PreProcessorHook to process the Mobile XML result from the service invocation. The Device's Transformer is applied to the result of the PreProcessorHook. If specified, the PostProcessorHook is invoked to process the markup page that is generated by the Device's Transformer.

10.6.2 Reference Model

This section describes the Oracle9iAS Wireless runtime, 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.

10.6.2.1 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 new request 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 starts the following sequence of activities within the runtime.

5. `createSessionId()`

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

6. `createSession()`

   Runtime creates a new Session for the given session id.

7. `registerSessionListeners(Request,Session)`

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

8. `beforeSession(SessionEvent)`

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

9. `findDeviceType(String)`

   Runtime dispatches to the DeviceIdentificationHook to determine the device model.

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. `getMobileId(Request,Session)`

    Runtime dispatches to the MobileIdHook to obtain the mobile 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. `sessionAuthenticated(SessionEvent)`

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

16. `getCurrentLocation(Request)`

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

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

    Runtime dispatches to the AuthorizationHook to authorize the requested service.

18. `serviceBegin(RequestEvent)`

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

19. `invoke(ServiceContext)`

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

20. `serviceEnd(RequestEvent)`

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

21. `transformBegin(RequestEvent)`

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

22. `process(Request,Element)`

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

23. `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.

24. `transform(Element,LogicalDevice)`

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

25. `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__.

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

    Runtime dispatches to the PostProcessorHook to process the device markup page to produce the final result.

27. `transformEnd(RequestEvent)`

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

28. `writeContent()`

    Runtime writes the content to the HTTPServletResponse.

29. `requestEnd(RequestEvent)`

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

30. `invalidate()`

    ParmImpl invalidates the Request object.

31. `afterRequest(RequestEvent)`

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

10.6.2.2 System Parameters

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

10.6.2.3 Static System Parameters

The Mobile XML results can contain the runtime variables, (composed from the names of the parameters) by appending two underscore characters (__) before and after the parameter name. These runtime variables in the Mobile XML results are "place holders" which are replaced by the values of the parameters during the post processing phase (Step 25 in Section 10.6.2.1, "Case: A Request Involving Session Establishment and Authentication") before the final result is returned to the requester. The following table 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. You can access them in one of two ways:

- Programmatically through the ServiceContext or the Request object:

  ```
  Arguments args = sc.getInputArguments();
  String language = args.getInputValue(X);
  ```

  where X is the parameter name.

- Through the PostProcessor for the final result 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 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:

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

10.6.2.4 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 10–11 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

10.6.2.5 General Guidelines for 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 wireless.jar file.

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

10.6.2.6 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() {   …. }
…
}
```

10.6.2.7 Compile Your Java Source

Make sure you have included the wireless.jar in your Java classpath during compilation.

10.6.2.8 Plug in Your Implementation through Property File

Set the corresponding entry in the **System Manager -> Site -> Wireless Web Server -> Event and Listeners** control panel, or the **System Manager -> Site -> Wireless Web Server -> Hooks** control panel to specify the name of the class that provides the implementation.

The following table lists the property entry name in the **System Manager -> Site -> Wireless Web Server -> Hooks** control panel for each hook.

Table 10–12 Property Entry Names for Hooks

Hook Name	Property Name
AuthenticationHook	wireless.http.locator.authentication.hook.class
AuthorizationHook	wireless.http.locator.authorization.hook.class

Table 10–12 Property Entry Names for Hooks

Hook Name	Property Name
CallerLocationHook	wireless.http.locator.caller.location.hook.class
DeviceIdentificationHook	wireless.http.locator.device.identification.hook.class
FolderRendererHook	wireless.http.locator.folder.render.hook.class
HomeFolderSorterHook	wireless.http.locator.home.folder.sorter.hook.class
ListenerRegistrationHook	wireless.http.locator.listener.registration.hook.class
LocationServiceVisibilityHook	wireless.http.locator.service.visibility.hook.class
PostProcessorHook	wireless.http.locator.post.processor.hook.class
PreProcessorHook	wireless.http.locator.pre.processor.hook.class
ServiceVisibilityHook	wireless.http.locator.service.visibility.hook.class
SessionIdHook	wireless.http.locator.session.id.hook.class
SignOnPagesHook	wireless.http.locator.signon.pages.hook.class
MobileIdHook	wireless.http.locator.mobile.id.hook.class
NormalizeAddressHook	wireless.http.locator.normalizeaddress.hook.class

For example, if you provide your own implementation of the authentication hook, you should set the wireless.http.locator.authentication.hook.class in the **System Manager -> Site -> Wireless Web Server -> Hooks** control panel to <your class name>.

10.6.2.9 Tips and Hints

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

10.6.2.10 Concurrent Requests

The Oracle9*i*AS Wireless 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.

10.6.2.11 Recursive Instances of Requests

The Oracle9*i*AS Wireless 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.

10.6.2.12 Query Parameters

The Oracle9*i*AS Wireless 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.

10.6.2.13 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.

10.6.2.14 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 Oracle9iAS Wireless runtime can be significantly degraded.

10.6.2.15 User-Defined Hooks Examples

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

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

10.6.2.16 Example 1

10.6.2.16.1 Changing the folder look and feel The look and feel of folders in Oracle9*i*AS Wireless can be changed in the following ways:

- Through configuration parameters which modify the built-in renderer
- By specifying a FolderRendererHook

- By specifying a FolderRendererService

10.6.2.16.2 Configuration parameters The look and feel of the folder can be changed by modifying the following configuration parameters:

Table 10–13 Configuration parameters

Name	type	default	Effect	Notes
wireless.device.login.enable	boolean	true	Display 'Login' link	Only displayed when user is not fully authenticated (guest or virtual user)
wireless.device.logout.enable	boolean	true	Display 'Logoff' link	Only displayed when user is fully authenticated (explicitly logged in)
wireless.device.userinfo.enable	boolean	true	Display 'Setup / User Info' link	
wireless.device.customizeservice.enable	boolean	true	Display 'Setup / Service' link	
wireless.device.globalpreset.enable	boolean	true	Display 'Setup / Presets' link	
wireless.device.userprofile.enable	boolean	true	Display 'Setup / User Profile' link	
wireless.device.register.enable	boolean	true	Display 'Register' link	Only displayed for guest or virtual user
wireless.device.help.enable	boolean	false	Display 'Help' link	Help page can be configured using the wireless.device.help.url
configuration parameter	wireless.device.home.enable	boolean	true	Display 'Home' link

10.6.2.16.3 FolderRendererService Oracle9*i*AS Wireless also allows an arbitrary service to be run when accessing a folder. This service is attached to the folder using the service designer; please see the service designer documentation for details. The service that renders the folder can either be active for that folder only, or for the given folder and all its children subfolders. The latter is useful for cases such as when one is customizing the folder look and feel for a subtree of folders. Customizing all user home folders is a prime example. If you put all user home folders beneath the folder /Users Home/, the FolderRendererService can then be

attached to the /Users Home/ folder, with recursive rendering turned on (see the webtool documentation for details on how to do this). If you want to have different folder rendering for different groups of users, you should group the users home folder under different group folders and attach different folder rendering services to each group folder, like this:

- /Portal1 -- attach folder render service 1
- /Portal2 -- attach folder render service 1

A folder service is written just like any other Oracle9*i*AS Wireless service, and will get invoked with a regular ServiceContext. The folder to be rendered can be retrieved using the ServiceContext method getCurrentFolder.

10.6.2.17 FolderRendererBean

The service used to render folders can be any Oracle9*i*AS Wireless service. It is usually convenient to write this service as a JSP, using the OC4J Adapter. In order to facilitate writing a FolderRenderer JSP service, the bean **oracle.panama.rt.hook.FolderRendererBean** is provided. This class has a number of methods for getting the content normally used by the built-in FolderRenderer: the getHeader, getBody and getFooter methods retrieve the header, body (folder content listing) and footer respectively. All methods in the FolderRenderBean takes a single argument, namely the current ServiceContext. In addition to the methods already mentioned, there are a number of utility methods (such as for getting the current user name), please see the FolderRendererBean JavaDoc for details.

The following example shows how write JSP code that displays a custom header, but reuses the built in folder renderer for displaying the folder content and footer:

```
<%@page import="oracle.panama.rt.ServiceContext"%>
<%@page import="oracle.panama.rt.hook.FolderRendererBean" %>

<%
    ServiceContext context = (ServiceContext)
        request.getAttribute("oracle.wireless.rt.context");
FolderRendererBean renderer =
         FolderRendererBean.getInstance();
response.setHeader("Mime-type", "text/xml");
%>
<SimpleResult>
  <SimpleContainer>
    <SimpleText>
      <SimpleTextItem>My custom header</SimpleTextItem>
```

```
    </SimpleText>
    <%= renderer.getBody(context) %>
    <%= renderer.getFooter(context) %>
  </SimpleContainer>
</SimpleResult>
```

10.6.2.18 FolderRendererHook

The third way of customizing the folder is by specifying a hook class that implements the interface **oracle.panama.rt.hook.FolderRendererHook**. This hook has a single method invoke, which takes as its argument the current ServiceContext and returns the DOM document containing the Mobile XML for the current folder. The hook will be invoked whenever there is no assigned folder rendering service.

10.6.2.19 FolderRendererPolicy

The default (built-in) implementation of the FolderRenderer is provided in the class **oracle.panama.rt.hook.FolderRendererPolicy.** This class can be subclassed, allowing custom hooks to reuse parts of the built-in functionality.

The main entry point for the FolderRendererHook is the **invoke** method. In the default FolderRendererPolicy implementation, the invoke method will create a **SimpleResult** element and in turn call **getHeader**, **getBody** and **getFooter** methods in order to append the header, body (folder content listing) and footer respectively. All methods in the FolderRendererPolicy takes a single argument, namely the current **ServiceContext.** If you need to add custom headers and footers, the Folder Renderer Policy can be subclassed to override the methods for getHeader and getFooter.

The following code is an example of a FolderRendererHook implementation that inserts a custom header:

```
import oracle.panama.rt.ServiceContext;
import oracle.panama.rt.hook.FolderRendererHook;
import oracle.panama.rt.hook.FolderRendererPolicy;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

class CustomFolderRenderer extends FolderRendererPolicy
implements FolderRendererHook {
    public Element getHeader(ServiceContext context) {
            Document doc = context.getXMLDocument();
            Element ret  = doc.createElement("SimpleText");
```

```
            Element text = doc.createElement("SimpleTextItem");
            ret.appendChild(text);
             String str = "My custom header";
            text.appendChild(doc.createTextNode(str));
            return ret;
    }
    // inherit getBody
    // inherit getFooter
}
```

10.6.2.19.1 Folder Setup Actions The default folder renderer in the runtime puts the controls for setting up the end user's preferences in the header and footer. The actions that are added in the device header/footer is described by the FolderSetupAction interface. When writing a folder rendering service or hook, it is possible to get information about all actions, including the URL (String), the localized label and whether the action should be displayed or not. Please see the FolderRendererBean and FolderRendererPolicy JavaDoc for a complete list of methods that retrieves FolderSetupActions.

Using the FolderSetupActions allows the user that extends the FolderRenderer to duplicate the built-in setup button semantics and labels, but substitute their own look and feel, for example by using SimpleHrefs instead of SimpleMenuItems. The following code is an example of a FolderRendererHook that does this:

```
import oracle.panama.rt.ServiceContext;
import oracle.panama.rt.hook.FolderRendererHook;
import oracle.panama.rt.hook.FolderRendererPolicy;
import oracle.panama.rt.hook.FolderSetupAction;

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

class CustomFolderRenderer extends FolderRendererPolicy implements
FolderRendererHook {
  public Element getHeader(ServiceContext context) {
    Document doc  = context.getXMLDocument();
    Element  ret  = doc.createElement("SimpleText");
    Element  text = doc.createElement("SimpleTextItem");
    ret.appendChild(text);
   FolderSetupAction[] actions = new FolderSetupAction[] {
     super.getEditPresetsAction(context),
     super.getEditServicesAction(context),
     super.getEditUserInfoAction(context),
```

```
    super.getLoginAction(context),
    super.getLogoffAction(context),
    super.getRegisterAction(context),
   };

  for(int i = 0; i < actions.length; i++) {
     if(actions[i].isActive(context)) {
       Element href = doc.createElement("SimpleHref");
      // set the URL of the href
       href.setAttribute("target", actions[i].getURL(context));
      // set the text to display for the href
       Text label =
         doc.createTextNode(actions[i].getLabel(context))
      href.appendChild(label);
text.appendChild(href);
    }
  }
  return ret;
 }

 // inherit getBody unchanged

 // override getFooter with implementation that creates footer
 // without setup buttons.
 public Element getFooter(ServiceContext context) {
   Document doc  = context.getXMLDocument();
   Element  ret  = doc.createElement("SimpleText");
   Element  text = doc.createElement("SimpleTextItem");
  text.appendChild(doc.createTextNode("My custom footer"));
  return ret;
 }
}
```

10.6.2.20 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 Oracle9iAS Wireless 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]
      }
}
```

```
}
```

10.6.2.21 Register the Authentication Hook

You should also add the name of the class, in this case hook.MyAuthenticator, in the System Manager > Site > Wireless Web Server > Hooks control panel in the Webtool under the wireless.http.locator.authentication.hook.class property.

10.6.2.22 Event Listener Example

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

10.6.2.23 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;

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
* @param    an event
**/
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()));
      event.getRequest().setAttribute(AttributeCategory.RUNTIME, SERVICE_END,
                       new Long(event.getTimeStamp()));
}

/**
* The event notification when transform begins
* @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.

10.6.2.24 Register the Request Listener

You should also add the name of the listener class, in this case listener.RequestListenerSample, in the System Manager > Site > Wireless Web Server > Event and Listeners control panel in the Webtool under the wireless.http.locator.request.listener.classes property.

10.6.2.25 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.

```
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;

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);
}

}
```

10.6.2.26 Register the Listener Registration Hook

You should also add the name of the listener registration class, in this case listener.ListenerRegistrationHookSample, in the System Manager > Site > Wireless Web Server > Hooks control panel in the Webtool under the wireless.http.locator.listener.registration.hook.class property.

10.6.2.27 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 Manager -> Site -> Wireless Web Server -> Event and Listeners** control panel of webtool.

10.6.3 Repository Data Model API

The Oracle9*i*AS Wireless Repository comprises the models for the Model-View-Control (MVC) architecture, while the Oracle9*i*AS Wireless runtime layer comprises the controllers for the MVC. The repository Model API in oracle.panama.model package lets you develop applications that create, delete, modify, and query the persistent objects in the Oracle9*i*AS Wireless Repository. Developers of custom adapters and transformers can implement the corresponding Model interfaces to develop the applications that supply the business processes and contents for the Oracle9*i*AS Wireless portal. The developers can also implement the "controller" applications, through the adapter, listener, or hook components, that manipulate the repository objects to perform provisioning, registration, personalization, accounting, and similar type of functions.

The Oracle9*i*AS Wireless repository imposes the organizational structure among the objects. For example, a User can belong to multiple Group's. The User is assigned one or more Role's. The user can access the Service's that are accessible to the groups to which the user belongs. However, the implementations of the User interface can access external provisioning systems or repositories, such as the Oracle Internet Directory (OID) and the Oracle Applications User Repository (AOL), to manage the information for the enterprise users and specify the user's roles, the user's group membership, and the particular services that are accessible to the user.

A Folder is a special kind of Service used as a container of the services to build the service trees. A Service or Folder can be assigned to one or more groups. The User can own a collection of DeviceAddresses, a collection of LocationMark's, a collection of customization Profile's, and one or more collections of Presets' which are used in advanced personalization. A default LocationMark and a default Profile can be assigned for each User. The Device interface in the Model API defines the

target device protocol (for example: WAP, SMS, or EMAIL), as well as specifies the physical characteristics of the target 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).

The intended users of the Model API are developers of customization portals, portlets, custom hooks, listeners, adapters, transformers, and applications such as JSPs, servlets, modules, and other (URL addressable) resources that are invoked through the HttpAdapter. Developers can also develop stand-alone applications which manipulate persistent objects using the Model API. Although these interfaces preserve the data integrity in the repository, they do not enforce access control security. The applications that access the repository through the Model API are not authenticated or authorized by the same Authentication and Authorization mechanisms in the Oracle9*i*AS Wireless runtime layer. In facts, the Model APIs are used by trusted components to develop and customize the authentication and authorization policies. The OracleMobile Online Studio, the System, Service, and Content Management Webtools, and the Customization Portals provide authentication and authorized access control to the repository. Developers should apply extreme caution when developing services using the interfaces in the Model API, and should take appropriate measures to prevent any undesired side effects when these services are invoked by the end users.

10.6.3.1 Data Model Cache and Synchronization

The repository objects are cached in the Java instances main memory when they are accessed from the Data Model API. These objects are removed from the main memory cache only after they are not accessed through the API for a time-to-live interval. This interval can be configured from "Cache Object Life Time" property in **System Manager -> Site -> Runtime Configuration** control panel in the webtool. If the repository object is modified and committed into the repository from one of the Java instances; all other Java instances will automatically reload the modified object from the repository. You can specify the number of cache synchronization threads from the **System Manager -> Site -> Object Cache Synchronization** control panel in the webtool.

10.6.3.2 Interfaces and Interface Hierarchy

The following sections describe the interfaces within the interface hierarchy in the Model API. These interfaces are contained in the `oracle.panama.model` package. For a sample application that illustrates the use of some of the interfaces, see Section 10.6.4.1, "Sample Code". The oracle.panama.model package also provides the following three locator and factory objects to access the model objects.

10.6.3.3 MetaLocator

MetaLocator, which is in the oracle.panama.model is used to access the ModelFactory and ModelServices.

10.6.3.4 ModelFactory

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

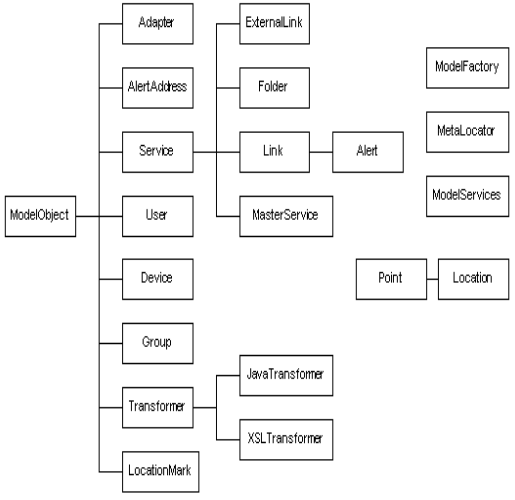
10.6.3.5 ModelServices

ModelServices, which is in the `oracle.panama.model` package, provides the locator or façade to access model objects.

10.6.3.6 ModelObject

The ModelObject is the root interface that represents the common behavior and properties of all repository objects. It is included in the `oracle.panama.model` package. The figure below illustrates the inheritance hierarchy among all of the interfaces in the oracle.panama.model package.

Figure 10–20 Model API Inheritance Hierarchy.



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

- Adapter
- Alert
- Community
- Device
- DeviceAddress
- ExternalLink
- Folder
- Group
- JavaTransformer
- Link
- Module
- LocationMark
- LocationPrivacyAuth
- MasterService
- Module
- PresetCategory
- PresetDescriptor
- Presets
- Profile
- Role
- Service
- Transformer
- User
- XSLTransformer

The following sections describe each subinterface.

10.6.3.7 Adapter

Adapter extends the ModelObject interface. Adapter is the repository container for the RuntimeAdapter, which is the interface that is to be implemented by all custom adapters. The Adapter incorporates the RuntimeAdapter classes into the repository and supports the loading and initialization of the RuntimeAdapter.

10.6.3.8 Device

Device extends the ModelObject 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 Nokia specific WML. Other examples are SMS and EMAIL. Device contains the Transformer objects.

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

10.6.3.9 DeviceAddress

DeviceAddress extends the ModelObject interface. DeviceAddress contains the device-specific address, such as a phone or an email address. The DeviceAddress takes precedence over the AlertAddress, which is deprecated in this release.

10.6.3.10 Group

Group extends the ModelObject 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.

10.6.3.11 LocationMark

LocationMark extends the ModelObject interface. It is a persistent object that represents the named and geocoded physical address.

10.6.3.12 PresetCategory

The PresetCategory extends the ModelObject interface. PresetCategory defines the structure and attributes of the Presets. Each PresetCategory contains a collection of PresetDescriptors, which provides the meta information for the attributes in the Presets relation.

10.6.3.13 PresetDescriptor

The PresetDescriptor extends the ModelObject interface. PresetDescriptor defines the meta data for an attribute in the Presets relation. The meta data of an attribute include the name, type, size, format, and description of the attribute.

10.6.3.14 Presets

The Presets interface extends the ModelObject interface. A Presets object contains a set of preset values whose structure and relation is defined by a PresetCategory. The Presets are owned by the User objects, and incorporates the personalized user preferences and frequently used input parameters for the services into the repository.

10.6.3.15 Profile

The Profile interface extends the ModelObject interface. The User can have one or more Profiles that encompass the user's customizations of the service trees. The Profile for a User can specify a preferred ordering of services in a folder.

10.6.3.16 Service

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

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.

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.

Module - Module extends the Service interface. A Module is a pointer to a "modulable" service with well known name called "virtual" URL. The modules could be local or remote.

Module - Module extends the Module interface. A Module is a pointer to a "modulable" local MasterService. Local MasterService means that it is in the same repository as the Module.

10.6.3.17 Transformer

Transformer extends the ModelObject interface. Transformer is the base interface for all transformation sub-classes. It is the repository container for the real transformation implementation (Java or XSL). It performs loading and initialization of the custom transformer classes that implements the oracle.panama.rt.xform.RtTransformer interface. It also provides the XSLT transformers for the XSLT stylesheets.

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. It incorporates the oracle.panama.xform.RtTransformer classes into the repository. It performs loading and initialization of the custom transformer classes that implements the oracle.panama.rt.xform.RtTransformer interface.
- **XSLTransformer** — XSLTransformer extends the Transformer interface. An XSLTransformer uses XSLT stylesheet which is expected to handle the transformation from the SimpleResult DTD to the device-specific markup language. It incorporates the custom XSLT stylesheets into the repository. It also provides the XSLT processors for the XSLT stylesheets.

10.6.3.18 User

The User interface extends the ModelObject interface. The User represents the identity of the user and facilitate personalization in the Oracle9*i*AS Wireless portals.

Each user can be assigned a private root folder to contain the user's personal quicklinks. The user can access the services in the groups to which the user belongs. The implementation of the User interface may access external provisioning system or enterprise repositories such as Oracle Internet Directory (OID) to manage the information about the user.

10.6.4 Sample Code that Uses the Data Model API

The following sample code illustrates how you can provision new objects into the Oracle9*i*AS Wireless repository using the interfaces in the Model API. We choose the standalone class to introduce the sample codes, although other type of components, such as adapters, hooks, listeners, and servlets can be used to illustrate the Model API. The example only shows the search, create, delete, and commit operations in the Model API but does not include the necessary business logics.

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.

- Use MetaLocator to get the ModelFactory and ModelServices (line **[1]**).
- Use ModelFactory to create a new object.
- Use ModelServices to search for an object.

```
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 singleton instance of this 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 interface name of the object. In this sample code, to create a new user (the code section for creating a new user starts in line [2]), you first look up the user by using the lookupUser(userName) method in the ModelServices interface (line **[3]**), as the following line of code shows:

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

Lookup operation should be the first step before creating any new persistent object in the Repository. The lookupUser(userName) method searches for the user by

name and, if the User by that name is found, returns the User object. If the user with that name cannot be found, the method throws the PanamaRuntimeException.

Next, you check if the group to which the user belongs (or should belong) already exists (line [4]). Following the convention 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 Group object. If the group is not found, the method throws the PanamaRuntimeException.

After checking if the user and the group already exist, you create the new user object (line **[5]** to line **[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 (line **[7]**):

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

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 if it is unable to save the work.

The searchUser() method in the sample code (line **[8]**) illustrates how to search a User object. To enumerate over a set of users (for example, all the users whose names start with the letter "B"), you use the ResultSetEnumeration (line **[9]**) returned by the method findUsers (line **[10]**). The method findUsers uses the pattern matching on the names. See also lines **[11]** and **[12]** in the listing of the complete sample code.

You should close the ResultSetEnumeration (line **[13]**) to release the database cursor, which otherwise will remain open.

To delete a User, you use the deleteUser method following the sample code section in line **[14]**. The user name must be exact in line **[15]**. ModelServices.lookupUser() method rejects the pattern matching templates by throwing exceptions. The user object is deleted in line **[16]**.

10.6.4.1 Sample Code

```
import java.util.Vector;

import oracle.panama.PanamaException;
import oracle.panama.PanamaRuntimeException;

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.model.Group;

/**
 * This is a sample program demonstrates the usage of the model API.
 */
public class SampleModelClient {

    private ModelFactory modelFactory;
    private ModelServices modelServices;

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

    /**
     * Get all group names
     */
    private String[] getGroupNames() throws PanamaException,
PanamaRuntimeException {
        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 ex;
        } finally {
            if (result != null) {
                result.close();
                result = null;
            }
        }
        return names;
    }

    /**
     * Create a new user.
     */
    private void createUser(String userName, String userPassword, String
groupName)    [2]
                            throws PanamaException, PanamaRuntimeException {
        try {
            // First check if the user does not already exists
            modelServices.lookupUser(userName);                    [3]
            // If we are here the user must already exists
            return;
        } catch (PanamaRuntimeException ignore) {}
        Group group = null;
        try {
            // Get the group to add the user
            group = modelServices.lookupGroup(groupName);          [4]
        } catch (PanamaRuntimeException ex) {
            // A PanamaRuntimeException is thrown if the group is not found
            group = null;
        }
        User user;
        // 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]
```

```
    // save the newly created object
    modelFactory.save();                                          [7]
}

/**
 * Search for users.
 */
private User[] searchUser(String userNamePattern)                 [8]
                        throws PanamaException, PanamaRuntimeException {
    User[] users;
    ResultSetEnumeration result = null;                           [9]
    try {
        result = modelServices.findUsers(userNamePattern);        [10]
        Vector buffer = new Vector();
        while (result.hasMoreElements()) {                        [11]
            User user = (User) result.next();                     [12]
            buffer.addElement(user);
        }
        users = new User[buffer.size()];
        buffer.copyInto(users);
    } catch (PanamaRuntimeException ex) {
        throw ex;
    } finally {
        if (result != null) {
            result.close();                                       [13]
            result = null;
        }
    }
    return users;
}

/**
 * Delete a user.
 */
private void deleteUser(String userName)                          [14]
                       throws PanamaException, PanamaRuntimeException {
    try {
        if (userName != null && userName.length() > 0) {
            User user = modelServices.lookupUser(userName);       [15]
            user.delete();                                        [16]

            // Save the changes
            modelFactory.save();
        }
    } catch (PanamaRuntimeException ex) {
```

```
            throw ex;
        }
    }

}
```

10.7 Adapters

Adapters are used to securely fetch application content and prepare it for device adaptation. Out-of-the-box, Oracle9*i*AS Wireless includes the HTTP Adapter. The HTTP Adapter is used to retrieve content from any HTTP / XML / J2EE server and application. The HTTP Adapter is compliant with HTTP 1.1. It supports HTTPS, cookies, and redirecting.

The method for creating mobile applications has been simplified in this release. Previously, it was common to create a Java Adapter for each mobile application. This would embed some of the application logic in an Adapter and some of the logic in the application itself. In order to leverage J2EE standards, the HTTP Adapter is recommended for mobile development. The complete mobile application can reside on any web server. The HTTP Adapter will point to the application URL to retrieve Oracle9*i*AS Wireless XML output. See the *XML Developer's Guide* section of this book for more information.

10.7.1 HTTP Adapter

The HTTP Adapter fetches the Mobile XML content from the external HTTP / HTTPS URLs. It acts as a proxy browser (which understands mobile xml) on behalf on the mobile device. Init Argument:

INVOKE LISTNER: This argument specifies the class path of the HTTP Adapter Listener. Refer to the javadoc of oracle.panama.adapter.http.event.HttpAdapterEventListener for more details on HttpAdapterEventListener Input Arguments.

Input Arguments:

1. URL: This argument specifies the URL to the data source
2. REPLACE_URL: This argument specifies whether the adapter should replace the relative URLs inside the fetched mobile xml document with absolute ones
3. FORM_METHOD: This argument specifies the HTTP method that should be used to open the data source URL

4. INPUT_ENCODING: This argument specifies the character encoding used by the adapter to send form parameters to the data source URL.

The HTTP adapter supports all the standard browser features:

1. Cookie Support: The HTTP Adapter implements the version 0 of the Cookie Specification by Netscape (http://www.netscape.com/newsref/std/cookie_spec.html). The HTTP Adapter stores the Cookies sent by the external URL's in the current session. And sends the relevant cookies (retrieved from the session) with the external HTTP URL request. The cookies are valid only for a session and are not stored persistently.
2. HTTPS Support: The HTTP Adapter can access https protocol based URL's. Before using https – the client certificates should be configured using the System Management Tool. Refer to the System Management Tool's documentation for more details.
3. Relative URL support: The Mobile XML returned by the external URL can use absolute or relative URL's as targets. The following mobile xml document uses both relative and absolute URL.

Example XML Document, showing the usage of relative and absolute URLs.

```
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1.0//EN"

"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
<SimpleContainer>
<SimpleHref target="http://Oracle9iAS
Wireless.oracle.com/HelloWorld.xml">Absolute URL</SimpleHref>
<SimpleHref target="HelloWorld.xml">Relative URL </SimpleHref>
</SimpleContainer>
</SimpleResult>
```

4. HTTP Adapter URL Prefix Configuration Parameter: If the Input argument URL doesn't start with http or https, then the value of the site configuration parameter "HTTP Adapter URL Prefix" is prepended to the value of input argument URL. Refer to the "Site Configuration" document to find more details on how to set the value of "HTTP Adapter URL Prefix" parameter.
5. HTTP Redirects: The HTTP Adapter honours the HTTP response code 301 to 305 and follows the redirected URL's, specified by HTTP Location header.
6. Post Redirect Support: The HTTP Adapter support post based redirects. To send a post based redirect the external application should send HTTP header

x-oracle-mobile-redirect with value **true,** and mobile xml form as the response content.

The following jsp file sends a Post redirect to the URL http://Oracle9*i*AS Wireless.oracle.com. The param1=value1 is passed as post data to the URL

```
<%
response.setHeader("x-oracle-mobile-redirect", "true");
response.setHeader("Content-Type", "tex/vnd.oracle.mobilexml");
%>
<?xml version = "1.0" encoding = "UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1.0//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<SimpleResult>
   <SimpleContainer>
      <SimpleForm name="ProcessSignOnForm" mimetype="text/vnd.oracle.mobilexml"
target="http://Oracle9iAS Wireless.oracle.com/MyApp" method="post">
         <SimpleFormItem name="param1" value="value1" type="hidden"/>
      </SimpleForm>
   </SimpleContainer>
</SimpleResult>
```

7. Proxy Server Support: HTTP Adapter can access external URL's through a HTTP proxy server. The proxy settings can be specified using the Site Configuration Tool.
8. Referring to non mobile XML documents: The HTTP Adapter rewrites all the targets specified in the mobile xml document so that they point to the HTTP Adapter. The mobile xml attribute "mimetype" can be used be indicate that the "target" points to a non-mobile xml document and should not be rewritten.
9. Support for GET and POST HTTP methods: HTTP Adapter uses the following logic to find the HTTP Request method to be used:
   - If the device sent a request through HTTP listener, then the method used by the device to send the Request to the Oracle9*i*AS Wireless server is used
   - Else if the input argument method has a non-null value, then the value of method is used
   - Else by default GET method is used.
10. Referral support: HTTP Adapter sends the HTTP Header Referer to specify the previous URL. This can be used by external applications to trace the context of the current request. By default, the Referer header is not sent, the mobile xml attribute "sendreferer" is used to indicate that the Referer header should be sent.

The following mobile xml document shows the usage of the sendreferer attribute.

```
<?xml version="1.0" encoding="UTF-8"?>
<SimpleResult>
<SimpleContainer>
<SimpleHref target="HelloWorld.xml" sendreferer=”true”>Send Referer</SimpleHref>
<SimpleHref target="HelloWorld.xml" sendreferer=”false”>Don't Send Referer
</SimpleHref>
</SimpleContainer>
</SimpleResult>
```

11. Device Information such as type of device and user Information like location, locale preferences etc. are passed as HTTP headers.

Following is the list of HTTP headers sent by the HTTP Headers.

Table 10–14 HTTP headers and their descriptions

Header Name	Description
x-oracle-user.locale	The locale preference of the User. For example, en-US
x-oracle-user.deviceid	The device identifier of the device.
x-oracle-user.userkind	The type of the User. Possible values are anonymous, virtual, registered
x-oracle-user.authkind	Whether is user is authenticated. Possible values are authenticated, unauthenticated
x-oracle-user.name	The name of the User. This header is sent only if the Disclose Identity option is selected by the user.
x-oracle-user.displayname	This display name of the User. This header is sent only if the Disclose Location option is selected by the user.
x-oracle-user.location.x	This header is sent only if the Disclose Location option is selected by the user.
x-oracle-user.location.y	This header is sent only if the Disclose Location option is selected by the user.
x-oracle-user.location.addressline1	
x-oracle-user.location.addressline2	
x-oracle-user.location.addresslastline	
x-oracle-user.location.block	
x-oracle-user.location.city	
x-oracle-user.location.county	
x-oracle-user.location.state	

Table 10–14 HTTP headers and their descriptions

Header Name	Description
x-oracle-user.location.postalcode	
x-oracle-user.location.postalcodeext	
x-oracle-user.location.country	
x-oracle-user.location.time	
x-oracle-user.location.type	
x-oracle-user.location.timesincelastupdate	
x-oracle-device.orientation	The orientation of the device. Possible values are landscape and portrait.
x-oracle-device.device	The type of device. Possible values are voice, microbrowser, pdabrowser, pcbrowser, micromessenger, messenger.
x-oracle-device.maxdocsize	The maximum size of the document (in bytes) that can be handled by the device.

The HTTP Adapter should be used to build mobile XML aware applications. The application can be built using any web programming technology like Java Server Pages (JSP), Servlet, Perl or Active Server Pages (ASP) and can be hosted on any web server. In Oracle9*i*AS Wireless 2.0 HTTP Adapter is the preferred way to build mobile xml applications.

10.7.2 Other Adapters

10.7.2.1 OC4J

The OC4J Adapter is used to fetch mobile xml content by invoking a JSP page in the same Java VM. The JSP page can access the request context information. The OC4J adapter is only for internal use of Oracle9*i*AS Wireless.

10.7.2.2 Web Integration

The Web Integration adapter retrieves and adapts Web content. The Web Integration adapter works with Web Interface Definition Language (WIDL) files to map source content to Portal-to-Go XML. Typically, the source format for the Web Integration adapter is HTML, but you can also use the adapter to retrieve content in other formats, such as XML. Portal-to-Go provides a visual tool for creating WIDL files, the Web Integration Developer. To create a WIDL file, you identify the elements of a Web page that you want to make accessible to a service. You then associate output

and input parameters to the source elements that you want to access in a Portal-to-Go service.

Note: The Web Integration adapter is deprecated in this release.

10.7.2.3 SQL Adapter

The SQL Adapter allows service designers to create services based on SQL Statements on Stored Procedures. Any database with JDBC driver is supported. The SQL Adapter uses pool of database connections. The connection pool parameters can be specified as init arguments of the adapter.

Note: The SQL adapter is deprecated in this release.

10.7.3 Creating Your Own Adapter

Customers can implement their own adapters by implementing `oracle.panama.adapter.RuntimeAdapter` interface (refer to javadoc). In this section we will implement a simple RMIAdapter, which fetches mobile xml content by invoking RMI methods.

Lets look at the implementation of the adapter

```
package oracle.panama.adapter.rmi;

import java.io.StringReader;

import java.util.Vector;
import java.util.Hashtable;
import java.util.Enumeration;

import java.lang.reflect.Method;
import java.lang.reflect.Member;
import java.lang.reflect.Modifier;
import java.lang.reflect.InvocationTargetException;

import java.net.MalformedURLException;

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.NotBoundException;

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

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

import oracle.panama.PAPrimitive;
import oracle.panama.Argument;
import oracle.panama.Arguments;
import oracle.panama.ArgumentType;
import oracle.panama.OutputArguments;
import oracle.panama.adapter.RuntimeAdapter;
import oracle.panama.adapter.RuntimeAdapterHelper;
import oracle.panama.adapter.AdapterException;

import oracle.panama.rt.ServiceContext;
import oracle.panama.core.xml.XML;

/**
  * A Simple RMI Adapter - invokes RMI methods to fetch mobile xml content
  */
```

All the adapters implement RuntimeAdapter interface

```
public class RMIAdapter implements RuntimeAdapter {

    // init arguments
    private Arguments initArgs = null;

    // input arguments
    private Arguments inputArgs = null;

    // output arguments
    private OutputArguments outputArgs = null;

    private boolean initialized = false;

    // reference to remote object
    private Object remoteObject = null;

    // remote interface
    private String remoteInterface;

    // hash table containing the method name to Method object mapping
    private Hashtable accessibleMethods = null;

    // Init argument - specifies the rmi url of the remote object
    private static final String RMI_OBJECT_URL = "RMI_OBJECT_URL";
```

```
    // Init argument - specifies the remote interface
    private static final String REMOTE_INTERFACE = "REMOTE_INTERFACE";

    // Input argument - specifies the remote method name to invoke
    private static final String METHOD_NAME = "METHOD_NAME";
```

The getInitArguments() method returns the init arguments required to initialize the adapter. The values of these arguments are specified during the creation of Master Service. The UI tools like Service Designer use this method of display the list of init that are required for creating a master service.

The RMI Adapter has following init arguments

- RMI Object URL: It specifies the URL of the remote object in the RMI name space.
- Remote Interface: It specifies the classpath of the remote interface

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

                    Argument arg = null;

                    arg = initArgs.createInput(RMI_OBJECT_URL);
                    arg.setComment("The RMI OBJECT URL for eg.,
rmi://rmiserver.com:2008/HelloWorld");
                    arg.setType(ArgumentType.SINGLE_LINE);
                    arg.setCaption("RMI Server URL");

                    arg = initArgs.createInput(REMOTE_INTERFACE);
                    arg.setComment("The Remote Interface");
                    arg.setType(ArgumentType.SINGLE_LINE);
                    arg.setCaption("Remote Interface");
                }
            }
        }
        return initArgs;
    }
```

This method returns the Input Arguments expected by the Adapter.

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

This method returns the Output Arguments

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

The init method initializes the adapter. The init adapter of the method is called once, when the master service pointing to the adapter is invoked or the getMergedInputArguments() method of the MasterService is called. The content of the init method must be synchronized to ensure that the class is not initialized by another thread.

The init method of RMIAdapter does the following

- Gets the value of init arguments (RMI_OBJECT_URL or REMOTE_ INTERFACE)
- Gets reference to remote object
- Inserts public methods of the remote interface in *accessibleMethods* hash table. The hash table is used later.
- Create input and output arguments of the adapter. The input arguments contain only one input argument METHOD_NAME of type enumeration.
- Sets initialized flag to true.

```
/**
 * 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) {
                    String rmiObjectUrl = args.getInputValue(RMI_OBJECT_URL);
                    remoteInterface = args.getInputValue(REMOTE_INTERFACE);

                    // check if both the init args are specified
                    if ((rmiObjectUrl == null) || (rmiObjectUrl.equals("")) ||
                        (remoteInterface == null) ||
(remoteInterface.equals(""))) {
                        throw new AdapterException("Init parameters missing");
                    }

                    // Get reference to remote object
                    Class interfaceClass = null;
                    try {
                        remoteObject = Naming.lookup(rmiObjectUrl);
                        interfaceClass = Class.forName(remoteInterface);
                    } catch (Exception ex) {
                        throw new AdapterException(ex);
                    }

                    Method[] methods = interfaceClass.getMethods();
                    accessibleMethods = new Hashtable();
                    for (int i=0; i<methods.length; i++) {
                        if (Modifier.isPublic(methods[i].getModifiers())) {
                            accessibleMethods.put(methods[i].getName(),
methods[i]);
                        }
                    }

                    // Create Input Arguments
                    inputArgs = RuntimeAdapterHelper.createArguments();
                    Argument arg = inputArgs.createInput(METHOD_NAME);
                    arg.setType(ArgumentType.ENUM);

                    String[] accessibleMethodNames = getAccessibleMethodNames();
                    arg.setOptions(accessibleMethodNames);

                    // Create Output Arguments
                    outputArgs = RuntimeAdapterHelper.createOutputArguments();
                    initialized = true;
                }
            }
        }
    }
```

The method returns an array of accessible method names

```
    // returns the array of accessible method names
    private String[] getAccessibleMethodNames() {
        Enumeration enum = accessibleMethods.keys();
        Vector v = new Vector();
        while (enum.hasMoreElements()) {
            String methodName = (String) enum.nextElement();
            v.add(methodName);
        }

        String[] methodNames = new String[v.size()];
        methodNames = (String []) v.toArray(methodNames);
        return methodNames;
    }
```

The invoke method is called when a client invokes a master service pointing to this adapter. The method executes the client request and returns the mobile xml result to the master service.

The method takes one argument of type ServiceContext. For each end user request received by the Oracle9*i*AS Wireless Server a ServiceContext object is created. The ServiceContext object contains all the user input arguments and arguments specified in Alias and Master Service.

```
/**
     * Invoke the adapter using the input and output parameters in the
     * service context.
     * @param serviceContext   the context that contains input parameters
     */
    public Element invoke(ServiceContext serviceContext) throws AdapterException
{
        checkState();
        // Get the input argument metod name
        String methodName =
                serviceContext.getInputArguments().getInputValue(METHOD_NAME);
```

If the method name is not specified the mobile xml displaying the list of available methods as menu items is returned

```
        if ((methodName == null) || "".equals(methodName)) {
            return getMethodMenuElement(serviceContext);
        } else {
```

The specified method is executed.

```
            return invokeRemoteMethod(methodName);
        }
    }
```

The destroy() method releases the resources acquired by the adapter in the init method.

```
    /**
     * Destroy the provider.
     */
    public void destroy() {
       remoteObject = null;
    }
```

Utility method to check if the adapter is initialized.

```
    private void checkState() throws AdapterException {
        if (initialized == false) {
            throw new AdapterException("Adapter is not initialized");
        }
    }
```

The invokeRemoteMethod() method invokes the remote method and converts the returned String to an XML DOM Element.

```
    private Element invokeRemoteMethod( String methodName) throws
AdapterException{
        Method method = (Method) accessibleMethods.get(methodName);
        if (method == null) {
            throw new AdapterException("method "+ methodName + " is not
available");
        }

        try {
            // invoke the remote method
            String retString = (String) method.invoke(remoteObject, new
Object[0]);
            Element elt = XML.makeElement(new StringReader(retString));
            return elt;
        } catch (Exception ex) {
            throw new AdapterException(ex);
        }
    }
```

The getMethodMenuElement() method returns mobile xml element for displaying available methods as menu items.

```
    private Element getMethodMenuElement(ServiceContext serviceContext) {
        // Returns SimpleMenu containing method names
        Document doc = serviceContext.getXMLDocument();
        Element simpleResultElt = PAPrimitive.createSimpleResult(doc, null);
        Element simpleContainerElt = PAPrimitive.createSimpleContainer(doc,
"MethodMenu");
```

```
        simpleResultElt.appendChild(simpleContainerElt);
        Element simpleMenuElt = PAPrimitive.createSimpleMenu(doc, "SimpleMenu");
        simpleContainerElt.appendChild(simpleMenuElt);

        Enumeration enum = accessibleMethods.keys();
        while (enum.hasMoreElements()) {
            String methodName = (String) enum.nextElement();
            Method m = (Method) accessibleMethods.get(methodName);
            String target =
RuntimeAdapterHelper.getURLPAoidParameter(serviceContext.getInputArguments());
            target += "&" + METHOD_NAME + "=" + m.getName();
            Element simpleMenuItemElt =
                PAPrimitive.createSimpleMenuItem(doc, m.getName(), target,
false);
            simpleMenuElt.appendChild(simpleMenuItemElt);
        }
        return simpleResultElt;
    }

}
```

The following sample RMI implementation can be used to test the RMI Adapter

SampleInterface.java: Remote Interface

```
package oracle.panama.adapter.rmi;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface SampleInterface extends Remote {
    // Returns Hello World message
    String sayHelloWorld() throws RemoteException;

    // Returns the current time
    String getTime() throws RemoteException;
}
```

SampleImpl.java: Remote Implementation

```
package oracle.panama.adapter.rmi;

import java.io.*;
import java.util.Calendar;

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
```

```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.registry.LocateRegistry;

import org.w3c.dom.*;
import oracle.xml.parser.v2.*;

public class SampleImpl extends UnicastRemoteObject implements SampleInterface {

    public final static int RMI_REGISTRY_PORT = 2099;

    public SampleImpl() throws RemoteException {
        super();
    }

    public String sayHelloWorld() {
        return createMobileXMLMessageString("Hello World!");
    }

    public String getTime() {
         return
createMobileXMLMessageString(Calendar.getInstance().getTime().toString());
    }

    String createMobileXMLMessageString(String message) {
        StringBuffer buf = new StringBuffer(1024);
        buf.append("<?xml version = \"1.0\" encoding = \"UTF-8\"
standalone=\"yes\" ?>");
        buf.append("<!DOCTYPE SimpleResult PUBLIC \"-//ORACLE//DTD SimpleResult
1.1.0//EN\" \"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd\">");
        buf.append("<SimpleResult>");
        buf.append("<SimpleContainer>");
        buf.append("<SimpleText>");
        buf.append("<SimpleTextItem>");
        buf.append(message);
        buf.append("</SimpleTextItem>");
        buf.append("</SimpleText>");
        buf.append("</SimpleContainer>");
        buf.append("</SimpleResult>");
        return buf.toString();
    }


    public static void main(String args[]) {

        // Create and install a security manager
```

```
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new RMISecurityManager());
        }
        try {
 // Create Registry
            LocateRegistry.createRegistry(RMI_REGISTRY_PORT);
            SampleImpl obj = new SampleImpl();

            // Bind this object instance to the name "HelloServer"
            Naming.rebind("//localhost:"+ RMI_REGISTRY_PORT+ "/Sample", obj);
            System.out.println("Sample bound in registry");
        } catch (Exception e) {
            System.out.println("Sample err: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Steps involved in testing the RMI Adapter

1. Create $ORACLE_ HOME/wireless/server/classes/oracle/panama/adapter/rmi directory
2. Compile RMIAdapter.java and copy the class file in $ORACLE_ HOME/wireless/server/classes/oracle/panama/adapter/rmi directory
3. Compile SampleInterface.java and SampleImpl.java
4. Generate RMI Stubs and Skeleton using rmic tool
5. Copy SampleInterface.class, SampleImpl.class, SampleImpl_Stub.class and SampleImpl_Skel.class files in $ORACLE_ HOME/wireless/server/classes/oracle/panama/adapter/rmi directory.
6. Load adapter into repository.
7. Create a Master Service pointing to this adapter.
8. Publish the Master Service.
9. Invoke the service from a phone simulator.

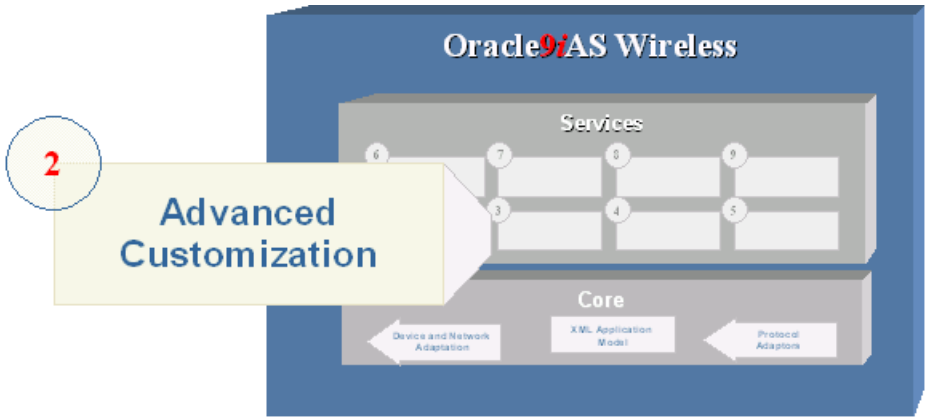
11

Advanced Customization

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

Oracle9*i*AS Wireless incorporates Advanced Customization that enables development of adaptable applications that personalize interactions and increase mobile application efficiency. The Advanced Customization allows for quick development and deployment of reliable, secure, scalable, and manageable applications. The end result is a complete one-to-one customer interaction.

Figure 11–1 Advanced customization

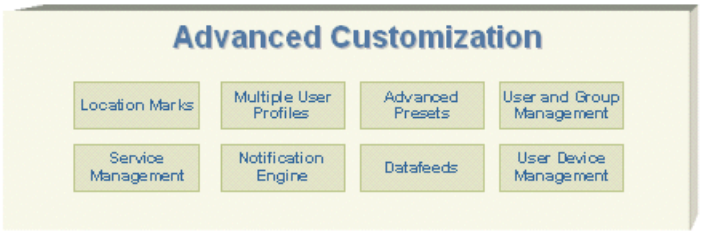


11.1 Overview of Advanced Customization

This chapter describes Oracle9*i*AS Wireless advanced customization features. Customization typically refers to how the user adapts the system or how the system adapts to the particular needs and preferences of the user. The user centric customization features give the users control over how they adapt the system to their needs and preferences. The system can also introduce mass customization technique that applies user profiling, sometimes by associating the user with like-minded group of users, to predict the user's needs and preferences, and adapt the system accordingly.

Customization is needed to make applications manageable by understanding visitors' needs based on their roles and preferences — for example, it is beneficial to present information in different ways to customers, suppliers, and employees. The ultimate goal is knowing enough about a customer's preferences and needs to intelligently suggest new services that they can use. The result is to turn a series of single transactions into a series of interactions that leads to an enduring, mutually profitable relationship.

Figure 11–2 Advanced Customization



Oracle9*i*AS Wireless includes a sample customization portal for PC browsers developed in Java Server Pages (JSP), which allows end users to customize the folder, service, bookmark, alert, alert address, location mark, and profiles. You can reuse these JSPs to re-brand the customization portal. You can also develop your own customization portals or integrate the customization tools to your existing portals. The developers should refer to the Runtime API, Data Model API, and the default JSPs for guidance when developing customization portals. You will find the concepts and features introduced in this chapter useful for designing the customization features to empower the end users.

You can introduce mass customization techniques using automatic user profiling. The usage history of the users can be found in the ptg_service_log and ptg_session_log tables in the Oracle9*i*AS Wireless repository. Some of the examples in this chapter describe how to extend the Oracle9*i*AS Wireless runtime to introduce mass customization.

The example customization portal lets the users manage the services, to rearrange, subscribe or unsubscribe, make bookmarks, and create quicklinks in each customization profile. Users can create new location marks and geocode the location marks. They can edit the presets that contain personal information or preferences. They can create new device addresses and valid them for alerts. They can subscribe to alerts for these devices.

End Users can log into the portal via a PC browser or any mobile devices to customize their Services.

The Customization 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. This gives a self branded Customization Portal.

Device-based Customization Portal targets the customers who want to customize their mobile services directly from small devices (such as cell phones, PDAs, etc.). The device Customization portal presents the services in edit mode and also

includes the user's preferences setting, Land Mark creation, Alert creation, Topic subscription in the menu list. Because of the limited size of display screen and the restricted input methods, the user interface and user interaction will be simplified to fit in those devices.

Multiple User Profiles make the mobile experience much more efficient and personalized. Users are able to manage their Customization Profiles from the Customization Portal. End Users may create, delete, modify and select the default Customization Profile.

Oracle9*i*AS Wireless gives the option to save the input values that a user has entered as a preset value for future invocations. Furthermore, Oracle9*i*AS Wireless gives options to enter a symbolic name to represent the presets. These symbolic names allow easy selection if there is more than one group of preset values. In addition, the user can manage their presets with the Customization Portal from any device or PC

Developers have the control to manage the types of presets available to particular users and groups and the input values used. Developers create Preset Categories for managing the presets that are available to end users.

Each Preset Category consists of a Preset Category Name (AddressForm for example) and any number of Preset Attributes. The first Preset Attribute is Street. For example, this will give end users the ability to create a Preset called Home, Work, etc. and each will have their own values for the Preset Attributes (Street, City...).

11.2 Presets

Oracle9*i*AS Wireless provides the facilities for the service developers and end users to apply extensive Customization to create personalized portals, which enhance the one-one relationships between the portal and each end user. One of the key Customization facilities is the Presets for storage of the user's personal information, preference settings, and frequently used input parameters on the server side so that the services can use them to generate the personalized responses.

The Oracle9*i*AS Wireless repository contains the concept of a portal User with predefined number of persistent attributes. These basic attributes include name, gender, date-of-birth, home postal address, country, language, default device address, etc. Presets are persistent objects in the Oracle9*i*AS Wireless repository that can be used to extend repository schema, especially to incorporate new persistent attributes for the User objects in the repository.

11.2.1 Presets Concept and Architecture

The Presets are persistent objects in the Oracle9*i*AS Wireless repository that can be used to extend the User schema and incorporate the user's personal information into the repository. Developers of the services can define the PresetCategories to extend the User schema in application-specific ways, for example to incorporate the billing address, credit card charge account, bank accounts, brokerage accounts, stock portfolios, emergency contacts, etc. These extended schemas may be defined and exclusively maintained by Personal Information Management (PIM) services.

The Presets can be also used to incorporate the user preferences into the repository. The user agent types and the logical device models in the repository describe the capabilities of the devices. Individual end users can customize some of the capabilities of the user agent. The Presets for user agent profiles can be used to let the end users customize the capabilities of the user agent, for example, to enable or disable sound, select background color, select quality of service, to disable images to minimize packet transmissions even though the device supports images, etc. The user agent profiles control the format of the content, but more general user preference profiles can affect the selection of the services and response of the services. For example, the user preference profile for sports, entertainment, technology, privacy requirements, etc. can be used by the services to filter the contents. The Presets architecture enables the development of adaptive web services based on the emerging Composite Capability / Preference Profile (CC / PP), User Agent profile (WAP UAProf), and Platform for Privacy Preferences (P3P) standards (*www.wapforum.org*).

The Presets can also store frequently used input parameters for the services. The services can define the attributes of the Presets relation to closely match the forms used by the services. These Presets can be used to auto fill the forms. The services can store the user inputs as the Presets for subsequent use. The Presets names uniquely identify the input parameter values and can be used as shorthand to significantly reduce the amount of data entry.

There are different categories of Presets in the repository. Each Presets relation contains a set of preset attribute values whose types and relations are defined by the PresetCategory. A User may own one or more Presets relations in each of the PresetCategory's. A PresetCategory contains a collection of PresetDescriptor's, each of which provides the metadata for the attributes in the Presets relation. The metadata of an attribute includes the name, type, size, format, and description of the attribute. For example, a Presets relation of the address book PresetCategory may contain the name, address, and phone number attributes of a contact for the user. Such a PresetCategory may be defined and exclusively maintained by a Personal Information Management (PIM) service. Another PresetCategory may

define the attributes of the Presets relations that contain the stock symbols, names, and classifications of the companies in the user's watch list or portfolio. The stock symbols in this category can be used as input parameters for the stock quote service.

The name of the PresetCategory must be unique within the repository. Likewise, the name of the PresetDescriptor must be unique within the PresetCategory to which it belongs. The name of the Presets relation is optional but if given the name must be unique among the Presets relations that are owned by the same User within the same PresetCategory. The PresetCategory's created programmatically are marked as system by default, i.e. they are to be maintained by the applications exclusively. System level PresetCategory's are not visible in the customization portals and cannot be edited by the end users directly. The applications can set the PresetCategory to non-system so that end users may edit its Presets in the customization portal.

11.2.2 Sample Applications

The PresetCategory's can be created programmatically as shown in the following examples. They can also be created from the **Service Designer > Preset Definitions** control panel in Webtool.

11.2.2.1 Example 1: Adding attributes to the User schema.

The following code fragment shows how to create a PresetCategory "Billing Address" to extend the User schema. The method first checks in line **[13]** if the "Billing Address" category already exists in the repository. If the category does not exist, the ModelFactory method createPresetCategory("Billing Address") is used to create the category in line **[21]**. Line **[23]** through **[27]** defines the first attribute "Addressee Name" of the category. Line **[25]** defines that the first attribute is comprised of a single line of text. In contrast, the second attribute "Street Address" is defined as a multi-line text field in line **[31]**. The new PresetCategory is committed in line **[47]**.

```
import oracle.panama.model.ModelFactory;
import oracle.panama.model.PresetCategory;
import oracle.panama.model.PresetDescriptor;
import oracle.panama.ArgumentType;
import oracle.panama.PanamaException;

public void createAddressBook() throws PanamaException {

        ModelFactory factory = MetaLocator.getInstance().getModelFactory();
        ModelServices services = MetaLocator.getInstance().getModelServices();
```

```
                    PresetCategory category;
                    try {
                        category = services.lookupPresetCategory("Billing Address");    [13]
                    } catch (PanamaRuntimeException ex) {
                        category = null;
                    }

                    if (category != null) {
                        return;   // category already exists
                    }
                     category = factory.createPresetCategory("Billing Address");        [21]

    PresetDescriptor descriptor = category.createPresetDescriptor("Addressee Name");   [23]
    descriptor.setDescription("The name of the addressee");
    descriptor.setPresetType(ArgumentType.SINGLE_LINE);   [25]
    descriptor.setStoredType(java.sql.Types.VARCHAR);
    descriptor.setSize(new Long(40));[27]

    descriptor = category.createPresetDescriptor("Street Address");
    descriptor.setDescription("The street address");
    descriptor.setPresetType(ArgumentType.MULTI_LINE);    [31]
    descriptor.setStoredType(java.sql.Types.VARCHAR);
    descriptor.setSize(new Long(120));

    descriptor = category.createPresetDescriptor("State");
    descriptor.setDescription("The name of the state");
    descriptor.setPresetType(ArgumentType.SINGLE_LINE);
    descriptor.setStoredType(java.sql.Types.VARCHAR);
    descriptor.setSize(new Long(2));

    descriptor = category.createPresetDescriptor("Zip code");
    descriptor.setDescription("The postal zip code");
    descriptor.setPresetType(ArgumentType.SINGLE_LINE);
    descriptor.setStoredType(java.sql.Types.NUMERIC);
    descriptor.setSize(new Long(5));

    factory.save();   [47]
}
```

Please note that the name of the PresetCategory must be unique in the repository. The createPresetCategory() method of the ModelFactory will throw oracle.panama.model.NameUniquenessViolationException if the application tries to create the PresetCategory with the same name. Likewise, the name of the PresetDescriptor must be unique within the PresetCategory. The createPresetDescriptor() method of the PresetCategory will throw oracle.panama.model.NameUniquenessViolationException if the application tries to

create the PresetDescriptor with the same name. The names of PresetCategory and PresetDescriptor are case sensitive and can contain any valid characters including spaces.

11.2.2.2 Example 2: Adding a unique Presets relation for the User

The following code fragment shows how the PresetCategory “Billing Address” is used to add persistent attributes to the User. If the "Billing Address" category does not exist, this method creates the new category. The example uses the unique object id of the User as the name of the Presets. The new Presets relation is created in line **[16]** only if the look up method in line **[13]** does not find any existing Presets relation with the same name. The use of the object id as the Presets name ensures that only one instance of the Presets relation for “Billing Address” is created for each user. The attribute values of the Presets relation are modified in line **[18]** through **[21]**. The modified Presets relation is committed into the repository in line **[23]**.

```
import oracle.panama.model.*;

public void addBillingAddress(User user, String addressee, String streetAddress,
                              String state, int zipCode) throws PanamaException {
        ModelFactory factory = MetaLocator.getInstance().getModelFactory();
        ModelServices services = MetaLocator.getInstance().getModelServices();

        PresetCategory category;
        try {
            category = services.lookupPresetCategory("Billing Address");
        } catch (PanamaRuntimeException ex) {
            createAddressBook();[9]
            category = services.lookupPresetCategory("Billing Address");
        }

        Presets presets = user.getPresets(category, Long.toString(user.getId()));[13]

        if (presets == null) {
            presets = user.createPresets(category, Long.toString(user.getId()));    [16]
        }

        presets.setPresetValue("Addressee Name", addressee);    [18]
        presets.setPresetValue("Street Address", streetAddress);
        presets.setPresetValue("State", state);
        presets.setPresetValue("Zip code", Integer.toString(zipCode));     [21]

        factory.save();   [23]
```

```
    }
```

Please note that the name of the Presets relation must be unique within the User's domain. If the application tries to create the Presets again with the same name for the same User, the createPresets() method of the User will throw the oracle.panama.model.NameUniquenessViolationException. The names of Presets relations are case sensitive and can contain any valid characters including spaces.

11.2.2.3 Example 3: Adding a unique Presets relation for Users' Profiles

The Profile's are repository objects that support multiple versions of personalized portals for each user. Let's suppose that the user has a Profile for "Business" and another Profile for "Personal" and requires a separate credit card charge account for each of the Profile's. The following code fragment describes how to create the "Credit Card Charge Account" category and the Presets relation that is unique for each profile of the user. The unique presets name is created from the object id of the User and the Profile in line **[43]** to ensure that only one Presets relation is created for each profile. The example also shows the use of preset type ArgumentType.ENUM for the "Card Type" attribute. The ENUM type lets you specify the valid options for that attribute as shown in line **[28]** and **[30]**. Lines **[60]** through **[62]** shows the use of the java.sql.Date type for persistent storage. The expiration date of the credit card is formatted using the java.text.DateFormat utility in line **[61]** so that it can be parsed and stored as Date type in the repository.

```
import oracle.panama.model.*;
import java.util.Date;
import java.text.DateFormat;

public void addCreditAccount(User user, Profile profile, String cardNumber,
                             String cardType, int expireMonth, int expireYear)
                             throws PanamaException {
        ModelFactory factory = MetaLocator.getInstance().getModelFactory();
        ModelServices services = MetaLocator.getInstance().getModelServices();

        PresetCategory category;
        try {
            category = services.lookupPresetCategory("Credit Card Charge Account");
        } catch (PanamaRuntimeException ex1) {
            try {
                category = factory.createPresetCategory("Credit Card Charge Account");
            } catch (PanamaException ex2) {
                throw ex2;
            }

            PresetDescriptor descriptor = category.createPresetDescriptor("Account Number");
```

```
            descriptor.setDescription("The credit card account number");
            descriptor.setPresetType(ArgumentType.SINGLE_LINE);
            descriptor.setStoredType(java.sql.Types.VARCHAR);
            descriptor.setSize(new Long(40));

            descriptor = category.createPresetDescriptor("Card Type");
            descriptor.setDescription("The type of credit card");
            descriptor.setPresetType(ArgumentType.ENUM);    [25]
            descriptor.setStoredType(java.sql.Types.VARCHAR);
            descriptor.setSize(new Long(40));
            String cardTypes[] = { "Master", "Visa", "Discover", "American Express", "Diners
Club" };      [28]
            try {
                descriptor.setOptions(cardTypes);     [30]
            } catch (TooManyOptionsException ex3) {
                throw new PanamaException(ex3);
            }

            descriptor = category.createPresetDescriptor("Expiration Date");
            descriptor.setDescription("The expiration date of the credit card");
            descriptor.setPresetType(ArgumentType.SINGLE_LINE);
            descriptor.setStoredType(java.sql.Types.DATE);

            factory.save();      [40]
        }

        String presetsName = Long.toString(user.getId()) + "-" + Long.toString(profile.getId());
[43]
        Presets presets = user.getPresets(category, presetsName);
        if (presets == null) {
            presets = user.createPresets(category, presetsName);
        }

        presets.setPresetValue("Account Number", cardNumber);
        presets.setPresetValue("Card Type", cardType);
        Date date = new Date(expireYear, expireMonth, 1);   [60]
        String dateStr = DateFormat.getInstance().format(date);    [61]
        presets.setPresetValue("Expiration Date", dateStr);    [62]

        factory.save();     [64]
    }
```

11.2.2.4 Example 4: Selecting the Presets relation under the current Profile.

The following fragment of codes from a RequestListener illustrates how the Presets relation for the "Credit Card Charge Account" is accessed during the serviceBegin() event notification. The routine throws AbortServiceException if no valid credit card charge account is available for the user. It checks for request profile, session profile, or default user profile in order as shown in line **[14]** and **[16]**. It composes the Presets name from the object id of the User and Profile. If the Presets relation for the "Credit Card Charge Account" is found, the listener provides the credit card information to the service as request parameters in line **[51]** through **[52]**.

```
import oracle.panama.rt.event.RequestEvent;
import oracle.panama.rt.event.AbortServiceException;
import oracle.panama.rt.Session;
import oracle.panama.rt.Request;

public void serviceBegin(RequestEvent event) throws AbortServiceException {
        Request request = event.getRequest();
        PresetCategory category;
        String presetsName;
        ModelServices services = MetaLocator.getInstance().getModelServices();
        String serviceName = request.getServicePath();
        User user;

        Profile profile = request.getProfile();      [14]
        if (profile == null) {
            profile = request.getSession().getProfile();    [16]
        }
        if (profile != null) {
            user = profile.getUser();
            presetsName = Long.toString(user.getId()) + "-" + Long.toString(profile.getId());
        } else {
            user = request.getSession().getUser();
            presetsName = Long.toString(user.getId());
        }

        try {
            category = services.lookupPresetCategory("Credit Card Charge Account");
        } catch (PanamaRuntimeException ex1) {
            throw new AbortServiceException("This service " + serviceName + " requires a valid
charge account");
        }

        Presets presets = user.getPresets(category, presetsName);
        if (presets == null) {
```

```
            throw new AbortServiceException("This service " + serviceName + " requires a valid
charge account");
        }

        String creditCardNumber;
        String cardType;
        String expiration;
        try {
            creditCardNumber = presets.getPresetValue("Account Number");
            cardType = presets.getPresetValue("Card Type");
            expiration = presets.getPresetValue("Expiration Date");
        } catch (PanamaException ex) {
            throw new AbortServiceException("This service " + serviceName + " requires a valid
charge account");
        }

        if (! creditAvailable(creditCardNumber, cardType, expiration)) {
            throw new AbortServiceException("This service " + serviceName + " requires a valid
charge account");
        }

        request.setParameter("Account Number", creditCardNumber);     [51]
        request.setParameter("Card Type", cardType);      [52]
        request.setParameter("Expiration Date", expiration);    [53]
    }
```

The above examples are based on the scenario that requires the applications to use well-defined naming conventions for the Presets relations, although the Presets names are optional. The following example illustrates a PresetCategory "Appointments" which allows multi-set entries. The identity of the Presets relation is provided by one of the attributes in the Presets relation. In this example, the Presets are created without names.

11.2.2.5 Example 5: Creating Presets without given name.

The following code fragment shows the PresetCategory "Appointments" that lets the users create appointment events. Since the attribute "Short Title" can be used to identify the events, the event Presets are created without names as shown in line **[65]**. All event Presets for the user can be retrieved from the repository as shown in line **[97]**. The "Appointments" category is set to non-system in line **[25]** so that the category can be included in the customization portal for end users to edit. The example shows the use of DateFormat utility to save the event time in line **[69]** and retrieve it in line **[105]**. The expired events are deleted from the repository in line **[112]**. The example also shows the use of the regular expression to constrain the

format of the "Phone Number" attribute. The regular expression is compatible with the public domain org.apache.regexp.RE toolset. The regular expression in line **[59]** is for the phone numbers in the US locale, which is

```
"\s*[(]?[1-9]\d{2}[)]?\s*-?\s*\d{3}\s*-?\s*\d{4}"
```

without the escape characters. The setPresetValue() method, line **[77]**, in the Presets will throw PanamaException if the value does not match the regular expression. The full regular expression syntax, which is compatible with the org.apache.regexp.RE toolset, is given in the next section.

```
import oracle.panama.model.*;
import oracle.panama.PanamaException;
import oracle.panama.PanamaRuntimeException;
import oracle.panama.ArgumentType;

import java.util.Vector;
import java.util.Enumeration;
import java.util.Date;
import java.text.DateFormat;
import java.text.ParseException;

public class SamplePresets {

    public void addAppointment(User user, String title, String memo, Date time,
                               boolean alarm, String phone) throws
PanamaException {
        ModelFactory factory = MetaLocator.getInstance().getModelFactory();
        ModelServices services = MetaLocator.getInstance().getModelServices();

        PresetCategory category;
        try {
            category = services.lookupPresetCategory("Appointments");
        } catch (PanamaRuntimeException ex1) {
            try {
                category = factory.createPresetCategory("Appointments");
                category.setSystem(false);         [25]
            } catch (PanamaException ex2) {
                throw ex2;
            }

           PresetDescriptor descriptor = category.createPresetDescriptor("Short
Title");
            descriptor.setDescription("Brief description of the event");
            descriptor.setPresetType(ArgumentType.SINGLE_LINE);
```

```
            descriptor.setStoredType(java.sql.Types.VARCHAR);
            descriptor.setSize(new Long(40));

            descriptor = category.createPresetDescriptor("Memo");
            descriptor.setDescription("Memo for the event");
            descriptor.setPresetType(ArgumentType.MULTI_LINE);
            descriptor.setStoredType(java.sql.Types.VARCHAR);
            descriptor.setSize(new Long(400));

            descriptor = category.createPresetDescriptor("Time");
            descriptor.setDescription("Time of event");
            descriptor.setPresetType(ArgumentType.SINGLE_LINE);
            descriptor.setStoredType(java.sql.Types.VARCHAR);
            descriptor.setSize(new Long(40));

            descriptor = category.createPresetDescriptor("Alarm");
            descriptor.setDescription("Enable or disable alarm before event");
            descriptor.setPresetType(ArgumentType.SINGLE_LINE);
            descriptor.setStoredType(java.sql.Types.VARCHAR);
            descriptor.setSize(new Long(1));

            descriptor = category.createPresetDescriptor("Phone Number");
            descriptor.setDescription("Optional phone number to ring for
alarm");
            descriptor.setPresetType(ArgumentType.SINGLE_LINE);
            descriptor.setStoredType(java.sql.Types.VARCHAR);
            descriptor.setSize(new Long(40));

descriptor.setFormat("\\s*[(]?[1-9]\\d{2}[)]?\\s*-?\\s*\\d{3}\\s*-?\\s*\\d{4}");
[59]
            descriptor.setEmptyOK(true);

            factory.save();
        }

        Presets presets = user.createPresets(category);    [65]

        presets.setPresetValue("Short Title", title);
        presets.setPresetValue("Memo", memo);
        String timeStr = DateFormat.getDateTimeInstance().format(time);   [69]
        presets.setPresetValue("Time", timeStr);
        if (alarm) {
            presets.setPresetValue("Alarm", "Y");
        } else {
            presets.setPresetValue("Alarm", "Y");
```

```
    }
   try {
        presets.setPresetValue("Phone Number", phone);          [77]
    } catch (PanamaException ex) {
        // ignore
    }

    factory.save();
}

public Presets[] getAppointments(User user) throws PanamaException {
    ModelFactory factory = MetaLocator.getInstance().getModelFactory();
    ModelServices services = MetaLocator.getInstance().getModelServices();

    PresetCategory category;
    try {
        category = services.lookupPresetCategory("Appointments");
    } catch (PanamaRuntimeException ex1) {
        throw new PanamaException(ex1);
    }

    Date now = new Date(System.currentTimeMillis());
    Vector allPresets = user.getAllPresets(category);       [97]
    Enumeration enum = allPresets.elements();
    Vector pending = new Vector();
    while (enum.hasMoreElements()) {
        Presets event = (Presets) enum.nextElement();
        String timeStr = event.getPresetValue("Time");
        Date time;
        try {
            time = DateFormat.getDateTimeInstance().parse(timeStr);   [105]
        } catch (ParseException ex) {
            time = null;
        }
        if (time != null && time.after(now)) {
            pending.add(event);
        } else {
            user.deletePresets(category, new Long(event.getId()));    [112]
        }
    }
    factory.save();

    Presets presetsArray[] = new Presets[pending.size()];
    pending.copyInto(presetsArray);
    return presetsArray;
```

```
    }
}
```

11.2.3 Regular Expressions Syntax for the Presets Attribute Formats

The full regular expression syntax that can be used to define for formats is:

Table 11–1 Characters

Character	Description
char	Matches any identical character
\	Used as an escape character (for example: *, \ \, \w)
\ \	Matches a single '\' character
\0nnn	Matches a character with given octet number
\xhh	Matches a character with given 8-bit hexadecimal value
\ \uhhhh	Matches a character with given 16-bit hexadecimal value
\t	Matches a tab character
\n	Matches a newline character
\r	Matches a return character
\f	Matches a form feed character

Table 11–2 Character Classes

Character	Description
[abc]	Simple character class
[a-zA-Z]	Range character class; range specified with "-" and "]" (for example: [x-z]
[^abc]	Negated character class, for exclusion tests.

Table 11–3 Standard POSIX Character Classes

Character	Description
[:alnum:]	Alphanumeric characters.
[:alpha:]	Alphabetic characters.

Table 11–3 Standard POSIX Character Classes

Character	Description
[:digit:]	Numeric characters.
[:upper:]	Upper-case alphabetic characters.
[:lower:]	Lower-case alphabetic characters.
[:space:]	Space characters (such as space, tab, and formfeed, to name a few).

Table 11–4 Variable Classes

Class	Description
.	Matches any character other than newline
\w	Matches an alphanumeric character
\W	Matches a non-alphanumeric character
\s	Matches a whitespace character
\S	Matches a non-whitespace character
\d	Matches a digit character
\D	Matches a non-digit character

Table 11–5 Boundary Matchers

Matcher	Description
^	Matches the beginning of a line
$	Matches the end of a line
\b	Matches a word boundary
\B	Matches a non-word boundary

Table 11–6 Greedy Closures (match as many elements as possible)

Element	Description
A*	Matches A 0 or more times (greedy)
A	Matches A 1 or more times (greedy)
A?	Matches A 1 or 0 times (greedy)

Table 11–6 Greedy Closures (match as many elements as possible)

Element	Description
A*	Matches A 0 or more times (greedy)
A{n}	Matches A exactly n times (greedy)
A{n,}	Matches A at least n times (greedy)
A{n,m}	Matches A at least n but not more than m times (greedy)

Table 11–7 Reluctant Closures (match as few elements as possible)

Element	Description
A*?	Matches A 0 or more times (reluctant)
A?	Matches A 1 or more times (reluctant)
A??	Matches A 0 or 1 times (reluctant)

Table 11–8 Logical Operators

Operator	Description
AB	Matches A followed by B (concatenation)
A | B	Matches either A or B(union)
(A)	Matches subexpression inside “(” and “)”, not including “(” and “)”

11.3 Location Marks

Location awareness is a key feature of Oracle9*i*AS Wireless. A user's location can be obtained from E911 or GPS units or Location Marks. Location Marks are user defined locations. For example, an end user may enter into their location-aware applications their home, work and headquarters office addresses. Then, when using a restaurant lookup application, the application can use the current location to provide driving directions. To ensure security and privacy, users can control which applications can access their location.

Due to the limitations of certain mobile devices such as telephones, it is difficult to input or display lengthy alphanumeric strings. A location mark stores a piece of spatial information identified by a concise, easy-to-understand name. For example, "My home" might be the name of a location mark, while the underlying spatial

information might be "123 Main Street, Somewhere City, CA, 12345; Lon = -122.42, Lat = 37.58".

Users have complete control of their location marks and are easily able to select, create, delete and modify location with any device or PC.

Location marks also allow users to try "what-if" scenarios: to make an application behave as if they were in a location different from their default or current location. For example, a user of an entertainment services application might be in Boston, but will be traveling to San Francisco in a few days. This person could set a location mark in San Francisco, and be presented with information relevant to the San Francisco area. Each user can have personalized location marks, which are stored in the Wireless repository.

Location marks are created using the LocationMark class. Users can also create location marks by logging into the Oracle9*i*AS Wireless Customization Portal, clicking the LocationMarks tab, and clicking Create. See Chapter 15, "Using Location Services" for more information on using Location Marks with Geocoding, Mapping, Routing, Traffic and Region Modeling services.

11.4 User Device Management

Oracle9*i*AS Wireless gives users, with multiple devices, the ability to easily manage and optimize their mobile experience for each device. The user can manage their devices from either a PC or mobile device. In addition, users are easily able to modify their current default device.

Once a user creates a new device profile, they can enter the following attributes for each device:

- Device Name
- Number of accepted alerts per day
- Address / number
- Device Type (voice, wap, pda...)

11.5 Multiple Customization Profiles

Oracle9*i*AS Wireless enables development of user-centric web services that adapt the contents not only to the device and network capability but also to the end user's preferences. The device portals typically provide the menu of services, which may be organized under several folders and subfolders. Menu driven device portals are

designed to optimize the interactive efficiency of wireless devices. Service menus are usually static but the portal may intelligently suggests new services to the user as it learns more about the user's needs and preferences. The Oracle9*i*AS Wireless server lets the end users personalize the portal by controlling the arrangement of the services in the menus. The portal can suggest new services to the user, but the user still controls when to include or exclude each service in the user's personalized portal. The administrators can explicitly prevent the end users from rearranging or removing certain services, such as promotions, preferred partners, emergency services, etc., from their personalized portals.

11.5.1 Concepts

The Profiles let the users create multiple personalized versions of the portals for their devices. The service menus may be different from one profile to another. For example, let's suppose that one of the folders for the user may contain the following five services:

- E-mail
- News
- Stocks
- Map
- Phone Directory
- Shopping

In the Home profile, the service menu in the folder may be customized as:

- Phone Directory
- E-mail
- News
- Stocks

The same folder may be customized differently for the Traveling profile as:

- Map
- Shopping
- Phone Directory

Multiple Profile's can be created for different roles, locations or contexts, device and network characteristics, or any other taxonomy. For a user with sale, marketing, and

consulting responsibilities who may play multiple roles in the enterprise portal, Profile's may be created for each of these roles to increase efficiency and accessibility of services. For a nomadic user who frequents among multiple metropolitan centers, the profiles may be created for each location. For example, a user's customization Profile for a cultural center like "San Francisco" may include services for theaters, sporting events, and BART schedules. The same user may have another Profile for the lake Tahoe area with a different combination of services. A location aware portal can automatically set the session Profile's for the users when they connect from different locations.

The Oracle9*i*AS Wireless runtime controllers can be extended to automatically provision the Profile's for users, for example to provide different views of the portal from more than one type of device. The example in the following section describes how to automatically provision a profile for the user. Alternatively, end users can create any number of Profile's for any context through the Customization Portal via a PC browser. Through the Service Management tool in the Customization Portal, they can customize the arrangement of services for each of the Profile's.

The administrators can specify the default sorting rules for the shared folders. Under the Profile architecture, end users can alter the default sorting rules to personalize the views of the shared folders. They can choose from the following five sorting rules:

- specified sequence numbers,
- lexicographic ordering,
- date of creation,
- frequency of access, and
- last access time.

The sequence numbers, lexicographic ordering, and date of creation produce the static views of the folders. Sorting by frequency of access or last access time produces the dynamic views of the folders. Furthermore, the administrators can control the static or dynamic arrangements of some of the services in the folder, such as emergency, promotion, and preferred partner's services, that may not be rearranged by the end users. The administrators can designate the segments of the views that may be rearranged or hidden by the end users. The view of the folder may be segmented such that one segment is sorted by the administrator's specification and another segment is sorted by the user's specification.

The Profile architecture lets the end users specify the visibility of a service in the profile provided the administrator does not explicitly disable the personalizable attribute of a service. This lets the end users "subscribe" or "unsubscribe" a service

that may be placed in the user's folder by the system. The system may also apply the location based filtering of the services in the location enabled folders, which offers additional dynamism to the views that vary with the user's mobile position.

Services that access the runtime objects can get the current Profile from the ServiceContext.getProfile method. See Section 10.6, "Runtime and Data Model APIs" for the description of runtime objects. This method first looks up the Profile in the current Request. If the Request does not specify a Profile, the method looks in the runtime Session for the session Profile. If the session Profile is empty, then the method looks up the default Profile of the User. This resolution strategy lets the Request overrides the session Profile, and the session to override the default User Profile. ServiceContext.getProfile can return null if there is no Profile at all. Applications should be prepared to react with default behavior when the Profile is not specified.

11.5.2 Sample Applications

The following example describes how to automatically provision a Profile for each of the devices that the User may use. The SampleRequestListener listens for the serviceBegin() event and provisions a new Profile in line **[25]** and **[27]** if the Request and Session do not already specify a Profile, line **[17]** and **[19]**. For the new Profile, it sets the user's home folder to sort the services in the home folder by the last access time of the service in line **[36]**. For each service that is view customizable in line **[39]**, it sets the service to be hidden in the Profile in line **[40]**. The end users can later customize the Profile's to unhide the services that they want to use. This needs to be done only once after the Profile's are first created. The listener then sets the Profile in the Request in line **[54]**.

```
import oracle.panama.model.*;
import oracle.panama.rt.Session;
import oracle.panama.rt.Request;
import oracle.panama.rt.event.RequestAdapter;
import oracle.panama.rt.event.RequestEvent;
import oracle.panama.rt.event.AbortServiceException;
import oracle.panama.PanamaException;

public class SampleRequestListener extends RequestAdapter {

    public void serviceBegin(RequestEvent event) throws AbortServiceException {
        Request request = event.getRequest();
        Session session = request.getSession();
        User user = session.getUser();

        Profile profile = request.getProfile();         [17]
```

```
        if (profile == null)
            profile = session.getProfile();                [19]
        if (profile == null) {
            Device device = request.getDevice();
            String deviceName = device.getName();
            Profile deviceProfile;
            synchronized(user) {
                deviceProfile = user.lookupProfile(deviceName);          [25]
                if (deviceProfile == null) {
                    deviceProfile = user.createProfile(deviceName);      [27]
                    ModelFactory factory =
MetaLocator.getInstance().getModelFactory();
                    try {
                        factory.save();
                    } catch (PanamaException ex) {
                        deviceProfile = null;
                    }
                    if (deviceProfile != null) {
                        boolean needCommit = false;
                        Folder home = user.getHomeFolder();
                        deviceProfile.setSortRule(home, SortRule.SORT_BY_ACCESS_
TIME_ASCEND);    [36]
                        Service[] services =
home.getAccessibleUserServices(user);
                        for (int i = 0; i < services.length; i++) {
                            if (services[i].isViewCustomizable()) {
[39]
                                deviceProfile.setHide(services[i], true);
[40]
                                needCommit = true;
                            }
                        }
                        try {
                            if (needCommit)
                                factory.save();
                        } catch (PanamaException ex) {

                        }
                    }
                }
            }
            if (deviceProfile != null)
                request.setProfile(deviceProfile);                [54]
        }
    }
```

```
}
```

11.6 User and Group Management

Oracle9*i*AS Wireless has advanced group and user management. Any user that is granted "User Manager" abilities is able to create, delete and modify groups and users through any device or PC. Any Group or User can be restricted or granted access to any folder or service.

Advanced Access Control List (ACL) used with the User and Group Management allows for fine grained access control that takes advantage of the flexible user or group policies that may be applied to Services or Folders

11.7 Service Management

Oracle9*i*AS Wireless offers complete control to the developer to manage what end users can do in terms of folder management. Developers can offer groups or users complete flexibility with their Service Management or restricted use of Service Management.

Services and folders may be organized in the following ways:

- user specified sequence numbers (any order)
- lexicographic ordering
- date of creation
- dynamic ordering based on frequency of access or last access

The end user also has the ability to customize their mobile experience with Bookmarks and Quicklinks. This gives users the ability to link frequently accessed services to the home deck or any other desired folder.

11.8 Rebranding the Customization Portal

11.8.1 Overview

Oracle9*i*AS Wireless Customization Portal is both a framework for the Customization 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 Customization 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 Customization Portal and their order of execution, as well as the file naming conventions and the directory structure used.

11.8.2 Page Naming Conventions

Some Customization 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 action / object combinations for all Customization Portal objects. For example:

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

11.8.3 JavaServer Pages Structure

Each Customization 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 11–3 Sample Page JavaBean Structure

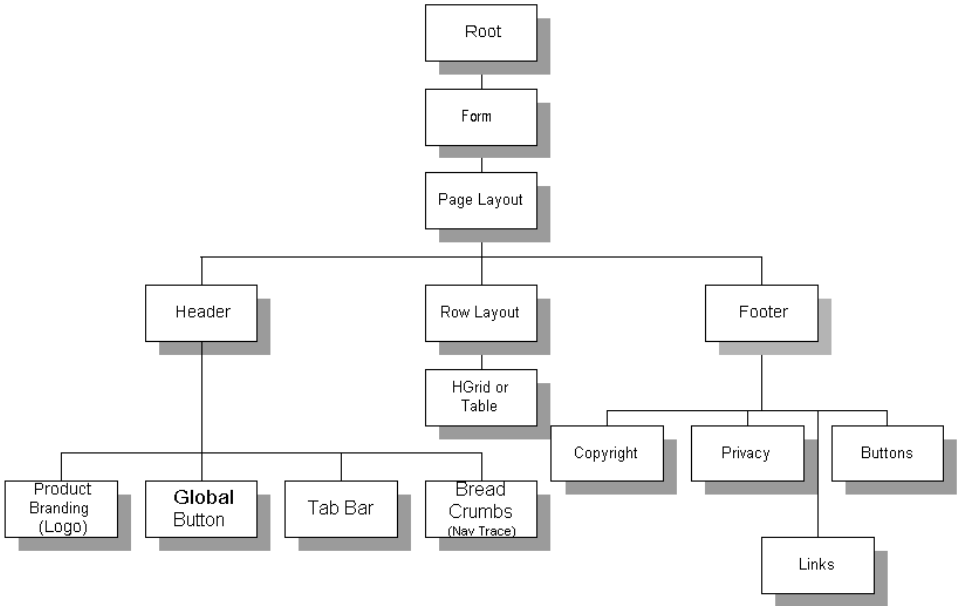


Figure 11–4 JavaBean Location on Sample Page

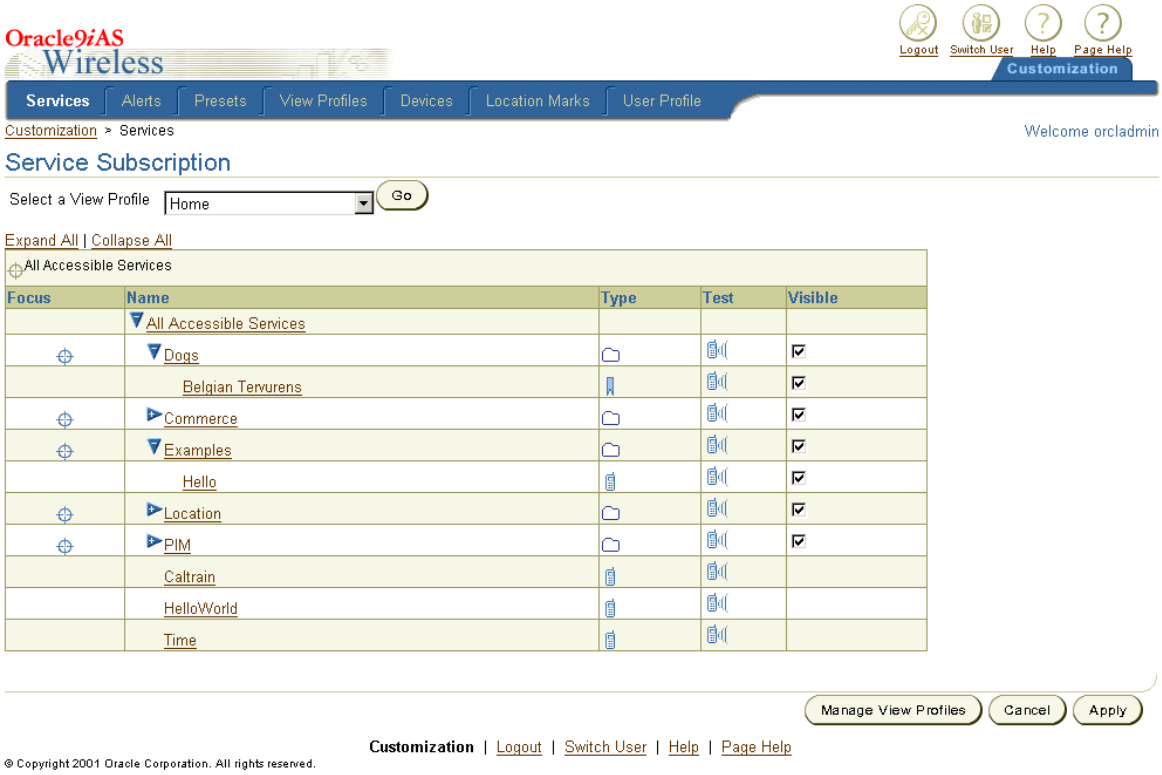


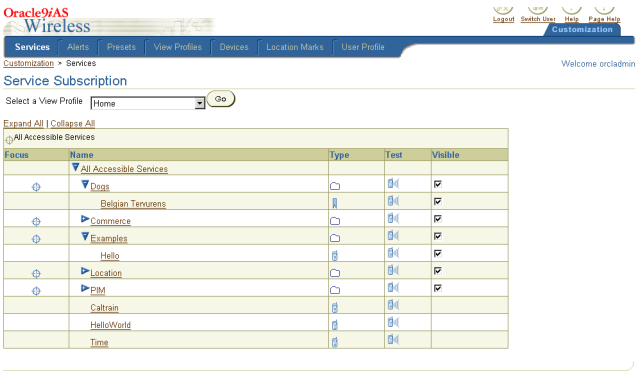
Table 11–9 JavaBean Function

JavaBean	Description
Root	Encompasses the JSP and allows it to be rendered as a whole.

Table 11–9 JavaBean Function

JavaBean	Description
1. Form and Page Layout	Establishes the Header and Footer and reserves the remainder of the page for other content. This component contains the Tab Bar, Navigation Trace, Row Layout, and Button elements.

Figure 11–5 Form and Page Layout



2. Header	Company branding and Tab Bar.

Figure 11–6 Header

3. Navigation Trace	Displays navigation cue and display name elements.

Figure 11–7 Navigation Tree

4. Footer	Global button links and Copyright information.

Figure 11–8 Footer

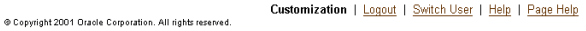
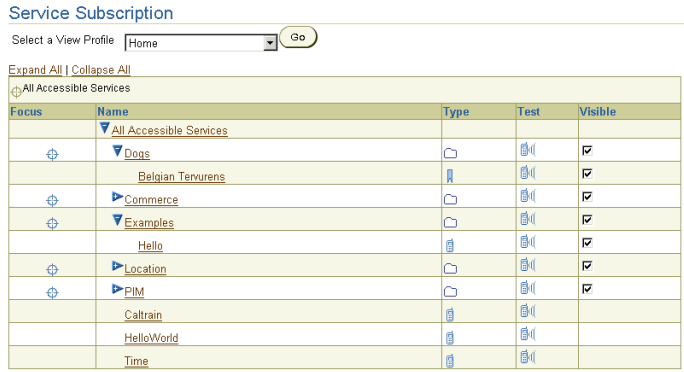


Table 11–9 JavaBean Function

JavaBean	Description
5. Row Layout	The main section of the page, which contains Tree, Cell, and Button elements.

Figure 11–9 Row Layout



6. HGrid Table — Linked hierarchical lists.

Figure 11–10 HGrid Table

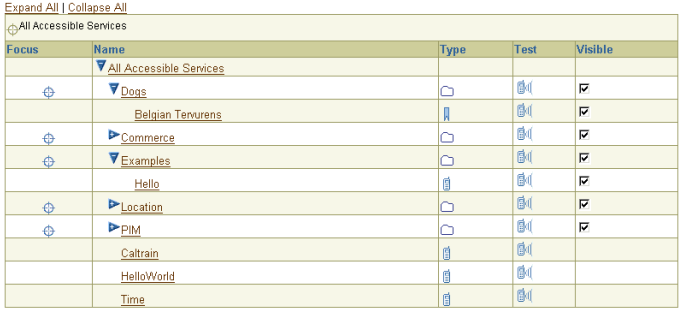


Table 11–9 JavaBean Function

JavaBean	Description
7. Button bar	Used to approve or cancel page actions. This is part of a row layout.

Figure 11–11 Button bar

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.

11.8.4 Directory Structure

To rebrand the Customization Portal, you modify the JSP files that generate the Customization Portal. After installing Oracle9*i*AS Wireless, these files are located in the $ORACLE_HOME/OC4J_Wireless/j2ee/applications/customization directory which has the following structure:

Table 11–10 Portal Directory Contents

Directory	Contents
customization-web	Container JSP files. Container files are accessed directly by browsers. also contains JavaScript files.
customization-web/images	Images used throughout the Customization Portal.
customization-web/Web-inf/jsp	Module JSP files. These files are included by either container JSP files or other module JSP files.
customization-web/cabo	JavaBean stylesheet, image, and JavaScript.
customization-web/cabo/images	JavaBean static images.
customization-web/cabo/images/cache	JavaBean generated images.
customization-web/cabo/jsLibs	Javascript.
customization-web/cabo/styles	Stylesheets.

Directory	Contents
customization-web / cabo / styles / cache	Generated stylesheets.
portal / messages / portal.properties	Sets page contents. For more information, see Section 11.8.7, "Setting the Multi-Byte Encoding for the Customization Portal".

11.8.5 Customization Levels

Customization 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.

11.8.5.1 Appearance Customization

This method requires replacing static strings in the HookFunc.jsp file located in the $ORACLE_HOME / OC4J_
Wireless / j2ee / applications / customization / customization-web / 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 11–11 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

11.8.5.2 Colors and Fonts

The colors and fonts can be customized by modifying the XML Style Sheet file:

`$ORACLE_HOME/j2ee/OC4J_Wireless/applications/customization/customization-web/`
cabo / styles / blaf.xss.

After the modification, remove:

`$ORACLE_HOME/j2ee/OC4J_Wireless/applications/customization/customization-web/`
cabo / styles / cache directory, and restart the server.

The new Colors and Fonts will take effect on the Web page.

11.8.5.3 JSP Modification

The JSP file PageTemp.jsp generates the Customization Portal page template. PageTemp.jsp is included in other JSP files which generate different contents in each page.

- PageTemp.jsp generates the logo and Tab bar at the top of the page.
- MyService.jsp presents a hierarchical view of the services available to the user.

11.8.6 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, Oracle9*i*AS Wireless first invokes EdFolder.jsp, then DoFolder.jsp.

Users can customize or configure the following:

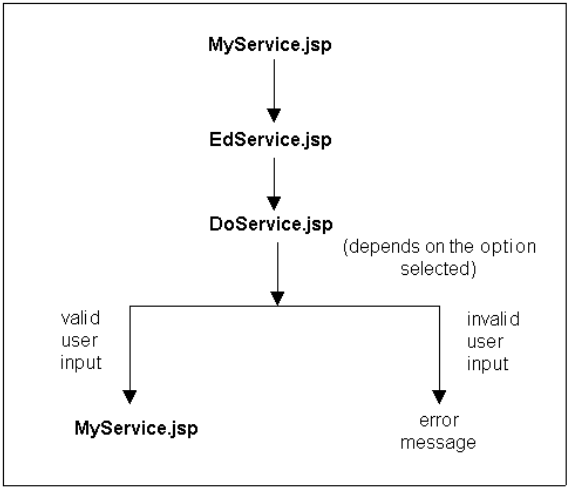
- Services
- Alerts(new alerts)
- Presets
- View Profiles
- Devices
- Location Marks
- User's Profile
- Deprecated (the following are only displayed if Deprecated Alert Support is set to TRUE [the default is FALSE])
- Alert (deprecated in this release)
- Alert Address (deprecated in this release)

11.8.6.1 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 11–12 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.

11.8.6.2 Creating New JSP

To create a new JSP, you implement the Customization Portal API. The classes in this API enable you to customize a version of the Customization Portal by providing a set of interfaces for portal customization.

11.8.7 Setting the Multi-Byte Encoding for the Customization Portal

The Customization 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 UTF-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.

The UI labels are loaded from portal_LANGUAGE (_COUNTRY if any). For example, portal_fr_CA.properties in the directory: $ORACLE_HOME/OC4J_Wireless/j2ee/server/classes/messages. Before login, the locale is determined by the Oracle9*i*AS Wireless locale setting. After login, the locale setting is determined by the user's locale preference.

11.9 Using the Customization Portal API

This section describes the Oracle9*i*AS Wireless Customization Portal API.

11.9.1 Overview

The Customization Portal API classes are designed to allow you to customize a version of the Oracle9*i*AS Wireless Customization 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 Customization 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 Customization Portal API, see the Oracle9*i*AS Wireless Javadoc.

11.9.2 Customization Portal API Classes

The Oracle9*i*AS Wireless Customization 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 Customization 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.

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.

11.9.2.1 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

For more details on Controller APIs, see the javadoc included with Oracle9*i*AS Wireless.

11.9.2.2 User Creation and Modification - UserController

UserController oversees operations involving users, such as:

- View and modify user profile
- View and manage users' views
- view and manage users' presets values

For more details on User Controller APIs, see the javadoc included with Oracle9*i*AS Wireless.

11.9.2.3 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.

For more details on Service Controller APIs, see the javadoc included with Oracle9*i*AS Wireless.

11.9.2.4 Alert Subscription Customization—Alert Subscription Controller

AlertSubscription handles operations of alert subscription and trigger settings such as:

- Create AlertSubscription
- Retrieve the trigger conditions of an alert
- Setting the values of trigger conditions of an AlertSubscription

For more details on Alert Subscription Controller APIs, see the javadoc included with Oracle9*i*AS Wireless.

11.9.2.5 Device Customization—Device Controller

Device Controller handles device customization activities, such as:

- Create and manage devices
- Validate and test devices
- WAP Provisioning

For more information, see *Oracle9iAS Wireless Getting Started and System Guide*, and the javadoc included with Oracle9*i*AS Wireless.

11.9.2.6 Alert Customization—AlertController (deprecated)

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.

> **Note:** This controller is deprecated in this release.

11.9.2.7 Locationmark Creation and Modification—LocationMarkController

LocationMarkController manages operations with locationmarks including:

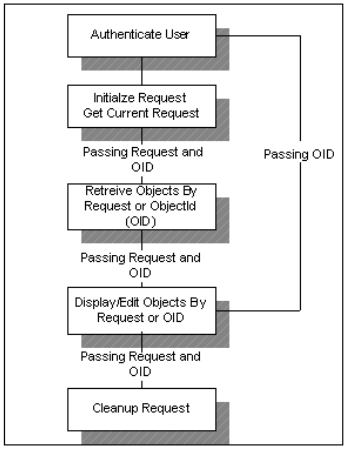
- Create a locationmark.
- View all the locationmarks.
- Modify a locationmark.
- Set default locationmark.
- Delete a locationmark.

For more details on LocationMark Controller APIs, see the javadoc included with Oracle9*i*AS Wireless.

11.9.3 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 11–13 Session Flow



11.9.3.1 Sample Code

The following code samples demonstrate an example of an HTTP session based on the process illustrated above.

11.9.3.2 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) {
}
```

11.9.3.3 Initialize Session

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

11.9.3.4 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) {
}
```

11.9.3.5 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) {
}
```

11.9.3.6 Cleanup Request

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

12

Alert Engine and Data Feeds

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

- Section 12.1, "Alert Engine"
- Section 12.2, "Data Feeders"

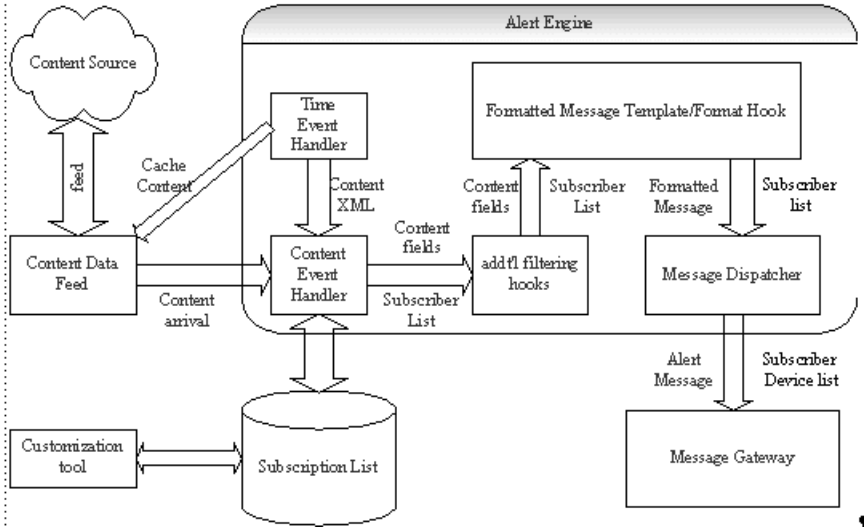
12.1 Alert Engine

The Oracle9*i*AS Wireless alert system provides you with extensible and scalable solutions to develop mobile alert services. An alert service generates alert message delivery events from a content source based on certain conditions such as a value-based predicate. For example, sending a stock quote to a user when the stock price has reached a predefined value. A condition can also be a time-based predicate with or without a value-based predicate. For example, a user can request the stock market index every day at 8 am, or a stock market index every day at 8 am, if the index has reached a predefined value. The delivery mechanism of the alert message is through Oracle9*i*AS Wireless message gateway, which allows notification to be sent through WAP push, Email, SMS messages, Instant Messaging or voice.

12.1.1 Alert Engine Architecture

The alert engine architecture is described in Figure 12–1.

Figure 12–1 Alert Engine Architecture



At design time, each master alert service selects a content descriptor, known as data feeder as its content source and becomes a subscriber to a content arrival event from that content source. Upon the content arrival event, which is generated by either data feeders or by custom applications, the alert engine notifies all the content event handlers whose associated master alert services have subscribed to the content arrival event. The content event handler in turn evaluates all the user alert subscriptions to locate the interested alert subscribers. Once the list of alert subscribers has been determined, the message formatter generates the outgoing alert message by either applying the message formatting template to the content or invoking the message formatter hook with the content and alert subscriber information. Finally, the alert message dispatcher communicates with the Oracle9*i*AS Wireless message gateway to deliver the message to the alert subscriber's device address.

The end user creates alert subscriptions using either the Oracle9*i*AS Wireless Content Manager or the Oracle9*i*AS Wireless alert subscription APIs. If a master alert service is set to be time base enabled, the user can specify the time and the frequency for the alert subscriptions to be evaluated. The time event handler requests the content from the data feeder based on the user time base setting and generates content arrival event for the content event handler.

12.1.2 Creating a Master Alert Service

When you create a master alert service, you (a service designer) must first describe the data content on which the alert service is built. Oracle9*i*AS Wireless enables you to define the data content using a data feeder. The data feeder module defines a given content in two forms: input parameters and output parameters. For example, a stock quote content can be defined as having a stock ticker as its input parameter and having price, volume, change, and change percentage as its output parameters. Furthermore, the data feeder module allows you to define the data mapping between any content source and the data feeder's input and output parameters so that contents can be retrieved automatically from any content source to Oracle9*i*AS Wireless.

12.1.2.1 Defining a Master Service

Defining a master service requires you to provide the following information:

- Which data feeder to use as the content descriptor
- Whether the master alert service allows users to create time-based predicate
- Which value trigger conditions can user defined on any of the output parameters of the selected data feeder
- How the notification message should be generated. There are two ways for the message to be generated:
  - Use a message text template to generate dynamic content.

    For example, if you are building an alert service based on a stock feed which has a stock ticker as its input parameter and price and change as its output parameters, you can define the message template as follows:

```
<SimpleText>
   Stock Alert for &ticker;
   Price:  &price;
   Change:  &change;
</SimpleText>
```

    You can refer to any data feeder input/output parameters in the template using the notation of ***&<parameter name>;*** , which is similar to the XML entity notation. When the notification message is generated at runtime, all references to the data feeder's parameter are replaced by the actual content values. In addition, the developer can use the following tokens in the template to generate a personalized message:

- **&USER;** -- The subscriber's account name
- **&USERDISPLAY;** -- The subscriber's display name
- Use an external Java hook for advance message formatting

  Developers wishing to generate more sophisticated notification messages with customized logic base on the runtime contents can supply a Java class which implements Java interface `AlertMessageFormatter` or `AlertPersonalMessageFormatter` for generic or personalized message respectively.

Note: Alert message device transcoding is not supported in current release of Oracle9*i*AS Wireless. However, the feature will be supported in the future releases

---

12.1.2.2 Extending the Alert Engine's Subscriber Filtering Capability

In some cases, the designer may wish to apply customized business logic before the notification message can be delivered to the subscriber. For example, some may want to interface with the billing system before any notification can be sent to the subscriber. Oracle9*i*AS Wireless alert engine allows the developer to supply a Java class which implements the Java interface AlertSubscriberFilter for additional subscriber filtering logic.

12.1.2.3 Using the Service Designer to Create Master Alert Service

Generally, you create a master alert service using the Webtool's Service Designer. The Service Designer provides you with a wizard that guides you through each step for creating a master alert service. For more information, see the Oracle9*i*AS Wireless Getting Started and System Guide.

12.1.2.4 Using the Java API to Create a Master Alert Service

The following code segment illustrates how to create a Master Alert Service using Oracle9*i*AS Wireless public APIs. This example shows how to use a stock quote feed and generate a stock alert for subscribers when a stock price reaches beyond or below the subscriber's predefined values.

```
MetaLocator m = MetaLocator.getInstance();
ModelFactory f = m.getModelFactory();
ModelServices s = m.getModelServices();
// use feed locator get the datafeeder information
```

```
DataFeeder df = s.lookupDataFeeder("StockFeed")
try {
// create a master alert with time base disabled
stockMS = f.createMasterAlertService("StockAlert", false, " Stock
Master Alert", df);
      stockMS.save();
    } catch (PanamaException pe) {
      System.out.println(pe.toString());
    }

    FeedMetaData fmdPrice = df.getOutputParameter("price");
              AlertConditionType actGT =
s.getAlertConditionTypeByName("GT");
    // add some conditions to the master alert
    AlertConditionMeta m1 = stockMS.addConditionDefinition("price_max",
fmdPrice, actGT, "30");
              AlertConditionType actLT =
s.getAlertConditionTypeByName("LT");
    AlertConditionMeta m2 = stockMS.addConditionDefinition("price_min",
fmdPrice, actLT, "15");

              // Set up alert message template
              StringBuffer messageTemplate = new StringBuffer("<SimpleText>
\n");
             messageTemplate.append("Stock Alert for &ticker; : \n");
             messageTemplate.append("Price: &price;  \n");
             messageTemplate.append("Change: &change; \n");
            messageTemplate.append("</SimpleText>");
             stockMS.setFormattedXMLTemplate(messageTemplate.toString());

// any create/update operation must be committed with the save. Alert object
do not use the
// wireless caching/persistence framework
    stockMS.save();

    // Lookup condition definitions
    AlertConditionMeta[] acm = stockMS.getConditionDefinitions();
    for (int i=0; i<acm.length; ++i) {
  System.out.println("Condition info : "+i+" id = "+acm[i].getId()+" Name =
"+acm[i].getConditionName());
    }

    // lookup input parameters. The input parameters get copied from
datafeeder.
// However the user can set default values
```

```
    AlertInputParamMeta[] aipm =
(AlertInputParamMeta[])mas.getInputParameters();

    for (int i=0; i<aipm.length; ++i) {
      System.out.println("Param info : "+i+" id = "+aipm[i].getId());
    }


   * On Section 12.1.10.2, in the code segment, please replace the following
     code:

try{
    eng.andleFeedContent("StockFeed",content);
}catch (PanamaException e){
      // handle the exception
   }

with the following new code segment:

try{
    eng.handleFeedContent("StockFeed",content);
}catch (PanamaException e){
      // handle the exception
   }
```

12.1.2.5 Publishing and Organizing Alert Services

Once the master alert service has been defined, the designer can create user-accessible alert services based on the master alert service. Oracle9*i*AS Wireless allows the designer to create and manage alert services through the following:

- Specifying captions on the trigger conditions defined in master alert service.
- Specifying default values on the input parameters defined in master alert service.
- Specifying default values on the trigger conditions defined in master alert service.
- Creating topics to organize alert services for optimum usage.
- Assigning the topic and alert services to any user group for access control.

12.1.3 Using the Content Manager to Create and Manager an Alert Service

Generally, you use the Webtool's Content Manager to create an alert service based on an existing master alert service. The Content manager guides you each step in creating an alert service. For more information on using the content manager, see the Oracle9*i*AS Wireless Getting Started and System Guide.

12.1.3.1 Use Java API to Create and Manage Alert Service

The following code segment illustrates how to create an alert service using Oracle9*i*AS Wireless public APIs. The use case is to create a stock alert service for master alert service *StockAlert* under a topic named *Stock Topic*. The alert service also sets the caption of the service for the trigger condition *price_max*, which is defined in the master alert service.

```
 MetaLocator m = MetaLocator.getInstance();

  ModelFactory f = m.getModelFactory();

  ModelServices s = m.getModelServices();


   Topic A = null;

   // create a topic named StockTopic associated with an alert service stock
alert
   try {

         A = f.createTopic("StockTopic", null);
        A.save();
         MasterAlertService mas = s.lookupMasterAlertService("StockAlert");
         AlertService as = f.createAlertService("stock alert ", mas);
          AlertConditionMeta[] conds = as.getConditionDefinitions();
         for ( index=0;index<conds.length; index++){
               if (conds[index].getConditionName().equals(“price_max”) ){
                    conds[index].setCaption(“above”);
                }
         }
         as.save();
         A.addService(as);
         A.save();

   } catch (Exception e) {
        L.e("createTopic failed :"+e.toString());
   }
```

12.1.4 Managing Alert Subscriptions

Once an alert service is created and assigned to a user group, the user belonging to the group can create or modify alert subscriptions as follows:

- Users can change content by specifying the input parameter values, such as stock ticker.
- The trigger condition values, such as the price value for an alert, to trigger when the stock quote reaches beyond a set point.
- The expiration time for a given alert subscription.
- Enable or disable an alert subscription.
- If the master alert service is set as a time-based enabled, the user can set the time and the frequency when the alert subscription should be evaluated. For a user can set this even for every week day at 8 am.

12.1.5 Managing Alert Subscription Using Customization

Users generally manage their alert subscriptions using the Wireless Customization. For more information, see the Oracle9*i*AS Wireless Getting Started and System Guide.

12.1.6 Manage Alert Subscription Using Java API

Managing alert subscriptions using Oracle9*i*AS Wireless public API is illustrated by way of the following example:

- A user, John, wants to create an stock alert subscription for stock ticker *ORCL* where he should be notified when the stock price reach beyond 50.
- In addition, John wants to be notified every day at 8 am if the condition mentioned above has been met.

Following code segment illustrates how to create an alert subscription for John using Oracle9*i*AS Wireless public APIs:

```
ModelServices s = m.getModelServices();

       AlertService as = s.lookupAlertService(“stock alert");

        //lookup user
       User u = m.lookupUser("John");

       UserAlertSubscription subs = as.addUserAlertSubscription(u);
```

```
        AlertInputParamValue[] apv = subs.getInputParameters();

         for(int i=0; apv!=null && i<apv.length; i++) {

      if(apv[i].getParamName().equals("ticker")) {
        apv[i].setValue("ORCL");
     }
         }

         // get/set new conditions for the subscription
         AlertConditionValue[] acv = subs.getConditions();

         for(int i=0; acv!=null && i<acv.length; i++) {

            if(acv[i].getConditionName().equals("price_max") ){
         acv[i].setValue("15");
                     break;
             }
          }

          subs.setHour(10);
          subs.setMinute(30);

           // get frequency from model services
           AlertTimeFrequency[] af = getAlertTimeFrequencies();
           for(int i=0; i<af.length; ++i) {
    if(af[i].getFrequencyCode().equals(AlertTimeFrequency.DAILY)){
      subs.setFrequency(af[i]);
    }
           }

    // any create/update operation must be committed with the save. Alert objects
              //  do not use the wireless caching/persistence framework
    subs.save();

      //... get all subscriptions
                 //retrieving subscsription
              UserAlertSubscription[] mysubs = as.getUserAlertSubscriptions(u);
              for (int i=0;i<mysubs.length;i++){
                 AlertInputParamValue[] params = mysubs[i].getInputParameters();
                 System.out.println("Sub ID: "+mysubs[i].getId());
                 System.out.print("Params:  ");
                 for (int j =0;j<params.length;j++){

  System.out.print(params[j].getParamType()+"="+params[j].getValue()+" ");
```

```
                    }
                    System.out.print("\n");
                    AlertConditionValue[] conds = mysubs[i].getConditions();
                    System.out.print("Conds:  ");
                    for (int j =0;j<conds.length;j++){

System.out.print(conds[j].getConditionName()+"="+conds[j].getValue()+" ");
                    }
                    System.out.print("\n");
                }
```

12.1.7 Creating a Device Address for Alert

A user must register a valid device to receive alert notifications. Furthermore, the user must select a device address for a given alert service. The device address must have the following attributes:

- Validity: The device address must be validated.
- Maximum number of alerts per day: The maximum number of alert messages that can be delivered to this device address per day.

For detail information regarding Device Address, please refer to the developer guide for Oracle9*i*AS Wireless core objects.

To set the user device address selection to a given alert service programmatically, use the API `setUserAlertDevice(DeviceAddress deviceAddress)` which is defined under interface `AlertService`.

12.1.8 Starting Alert Engine Process

To deploy a master alert service, a system manager creates (or updates) an alert instance, adds the newly created master alert service to the alert instance and then starts the alert instance. See the *Oracle9iAS Wireless Getting Started and System Guide* for more information managing the Oracle9*i*AS Wireless system and starting and stopping an alert engine process.

Note: Oracle9*i*AS Wireless alert engine uses Oracle9*i*AS Wireless message gateway system to deliver message to device addresses. The transport must be up and running with the selected drivers for different delivery types.

12.1.9 Notifying the Alert Engine for Content Arrival

This section describes the methods of pushing contents to the Oracle9*i*AS Wireless alert engine for notification delivery. These methods include:

- The Data Feeder Module
- The Alert Engine Java API

12.1.9.1 Data Feeder Module

If a designer of an master alert service chooses to use the Oracle9*i*AS Wireless data feeder module to retrieve data content, also known as the "POLL" model, the data feeder automatically notifies the alert engine of the content arrival event. The alert engine then evaluates all the user subscriptions with the newly arrived content to deliver alert message to the appropriate subscriber device addresses.

12.1.9.2 Alert Engine Java API

If the designer chooses to use the "PUSH" model by defining a pass-through data feeder, or the designer wishes to deliver contents to the alert engine programmatically from an application, the designer uses the alert engine public APIs to notify alert engine of the content arrival event.

The following code segment illustrates how to notify alert engine using Java APIs:

```
AlertEngine eng = AlertEngineLocator.getInstance().getAlertEngine();
Hashtable content = new Hashtable();

content.put("ticker","ORCL");
content.put("price","16");
content.put("change","1");

try{
    eng.andleFeedContent("StockFeed",content);
}catch (PanamaException e){
      // handle the exception
```

12.2 Data Feeders

The Data Feeder is the agent that downloads content. The data feeder runs periodically, independently of service invocations. The feed framework is designed to download content for a Oracle9*i*AS Wireless process. The downloaded content can be used both for asynchronous alerts as well as cached data for synchronous services.

The download schedule for the data feeder is maintained in the update policy for that data feeder. The update policy determines the update interval, or how often the data feeder runs. The update policy can the time of day, and which days of the week to run the data feeder.

Each data feeder has a content provider, which is the source of the content. The content provider maintains information about the URI of the content, the protocol to use for downloading the content, and the format of the data to be downloaded.

When specifying a feed, the user sets up a metadata definition of the content to be downloaded using feed parameters. These parameters are instances of the data type, `FeedMetaData`. Feed parameters have an underlying SQL data type chosen from a predefined set of types, defined in `oracle.panama.feed.FeedUtil`.

Feed Input Parameters are input parameters particular to a content provider. They specify the data used when requesting data from the content provider. For example, when downloading data from a content provider using HTTP, the input parameters will be used either to construct a GET URL or as POST parameters in the HTTP request.

Feed Output Parameters define the data type of the output from the content provider.

The runtime behavior of the data feeder can be customized with the `FeedDownloadHook` and the `FeedDataFilterHook`.

The `FeedDownloadHook` is used to customize the URI used when downloading content. For example, in a HTTP download, the input parameters are, by default, used to construct a GET URL, with the input parameters used as GET HTTP parameters. In some cases, however, the base URL depends on the input parameters. In such a case, the URL would be http://www.ahost.com/input_param_1/input_param2/index.html. The behavior for constructing the URL can be overridden with a custom `FeedDownloadHook` to achieve the desired result.

The `FeedDataFilterHook` is used to do additional processing on the downloaded content. As each row of data is downloaded, the data filter hook gets invoked on each row. This allows the feed implementer to perform special processing, such as splitting a single output parameter into several output parameters.

The pass-through data feeder is a datafeeder that accesses local content through user-defined Java code. Consequently, a pass-through data feeder has neither a content Provider nor an update policy. Similarly, the `FeedDownloadHook` and the `FeedDataFilterhook` are not relevant for a pass-through data feeder. The feed metadata still needs to be set up for a pass-through data feeder.

Note: In Oracle9*i*AS Wireless Version 2, the feed framework is designed perform request-reply (data pull feeds). Although the architecture has been designed to accommodate push data feeds, this functionality is not included in this version.

12.2.1 Building a Data Feeder

You can create a data feeder using the Webtools Service designer, or programatically.

Creating a data feeder can be broken into the following steps:

1. **Create a named data feeder:** all data feeders must have a name. The name may be changed. The data feeder also has a object ID, which is permanent and unique.
2. **Set Content Provider parameters:** set the protocol and format for the current Content Provider. There are constants for the built-in protocols and formats.
3. **Create Data Feeder Input parameters:** A data feeder must have at least one input parameter. For each input parameter you specify, you must give an internal name and data type. Parameters may have options that depend on the chosen format. If the format chosen is delimited text, you have the option of specifying the column number in which the input parameter appears. This is useful if the input parameter is also included in the output from the content provider. The index for the columns starts at 1, as SQL. If 0 is specified, then the input parameter is assumed to not be in the output.
4. **Create Data Feeder Output parameters:** A data feeder must have at least one output parameter. The output parameter can be customized in the same manner as an input parameter.
5. **Finalize the Feed**: Finally, you must call the DataFeeder method `createFeedDefinition`. This method creates the feed metadata definition in the repository, which is required to use the feed and the feed cache table. Once the feed definition has been created, feed parameters cannot be deleted, only renamed.

the Webtool's Service Designer provides you with a wizard to guide you through each step of the creation process. For more information, see the *Oracle9iAS Wireless Getting Started and System Guide*.

12.2.2 Creating a Passthrough DataFeeder

A pass-through datafeeder requires that you specify the classname of the pass-through datafeeder to use. It does not require all the information that a regular datafeeder needs: in particular, the protocol and format to use is irrelevant.

The following code creates a pass-through datafeeder:

```
ModelFactory mf = MetaLocator.getInstance().getModelFactory();
        // Create a named datafeeder
        PassthroughDataFeeder df = mf. createPassThroughDataFeeder ("stock_passthrough")
        // Set the class name to use for implementation
        df.setClassName("fully.qualified.package.and.Class");
        // Create input parameters
        FeedMetaData fmi = df.createMetaData("sym", "TEXT_30");
        df.addInputParameter(fmi):
        // Create output parameters
        FeedMetaData fmo1 = df.createMetaData("price", "NUMBER");
        df.addOutputParameter(fmo1);

        FeedMetaData fmo2 = df.createMetaData("change", "NUMBER");
        df.addOutputParameter(fmo2);

        // Finalize the feed -- create feed definition
        // in repository
        df.createFeedDefinition();
```

12.2.3 Sample Applications

12.2.3.1 Sample Application: Downloading Stock Quotes in XML

Oracle9*i*AS Wireless includes the **sample200.xml**. This sample file contains a datafeeder for retrieving stock quotes over HTTP. The stock quotes are in XML format; the sample datafeeder includes the stylesheet for extracting the relevant values from the XML input feed.

In order to create this data feeder programmatically, you would use the following code:

```
ModelFactory mf = MetaLocator.getInstance().getModelFactory();
// Create a named datafeeder
DataFeeder df = mf.createDataFeeder("stock_screamingmedia");
// Set content provider parameters
ContentProviderInfo cpi = df.getContentProviderInfo();
cpi.setProtoolType(ContentProviderInfo.PROTOCOL_HTTP);
cpi.setPrimarySource("http://www.screamingmedia.com/");
```

```
cpi.setFormatType(ContentProviderInfo.FORMAT_XML);
// Create input parameters
FeedMetaData fmi = df.createMetaData("sym", "TEXT_30");
df.addInputParameter(fmi):
// Set the parameters for this parameter and content provider
Map paramOptions = new Hashtable();
paramOptions.put(ContentProviderInfo.COLUMN_NUMBER, 1);
cpi.setParamArguments(fmi, paramOptions);


// Create output parameters
FeedMetaData fmo1 = df.createMetaData("price", "NUMBER");
df.addOutputParameter(fmo1);
paramOptions.put(ContentProviderInfo.COLUMN_NUMBER, 2);
cpi.setParamArguments(fmo1, paramOptions);

FeedMetaData fmo2 = df.createMetaData("change", "NUMBER");
df.addOutputParameter(fmo2);
paramOptions.put(ContentProviderInfo.COLUMN_NUMBER, 3);
cpi.setParamArguments(fmo2, paramOptions);
// Finalize the feed -- create feed definition in repository
// create cache table as needed
df.createFeedDefinition();
```

12.2.3.2 Sample Application: Downloading Stock Quotes in CSV Format

sample200.xml. also includes a datafeeder for retrieving stock quotes over HTTP that downloads the stocks in the comma-separated variable (CSV) format.

The following code illustrates how to create this data feeder programmatically.

```
ModelFactory mf = MetaLocator.getInstance().getModelFactory();
// Create a named datafeeder
DataFeeder df = mf.createDataFeeder("stock_yahoo")
// Set content provider parameters
ContentProviderInfo cpi = df.getContentProviderInfo();
cpi.setProtocolType(ContentProviderInfo.PROTOCOL_HTTP);
cpi.setPrimarySource("http://quotes.yahoo.com/quote");
cpi.setFormatType(ContentProviderInfo.FORMAT_DELIMITED);
// Create input parameters
FeedMetaData fmi = df.createMetaData("sym", "TEXT_30");
df.addInputParameter(fmi):
// Set the parameters for this parameter and
// content provider
Map paramOptions = new Hashtable();
```

```
paramOptions.put(ContentProviderInfo.COLUMN_NUMBER, 1);
cpi.setParamArguments(fmi, paramOptions);

// Create output parameters
FeedMetaData fmo1 = df.createMetaData("price", "NUMBER");
df.addOutputParameter(fmo1);
paramOptions.put(ContentProviderInfo.COLUMN_NUMBER, 2);
cpi.setParamArguments(fmo1, paramOptions);

FeedMetaData fmo2 = df.createMetaData("change", "NUMBER");
df.addOutputParameter(fmo2);
paramOptions.put(ContentProviderInfo.COLUMN_NUMBER, 3);
cpi.setParamArguments(fmo2, paramOptions);

// Finalize the feed -- create feed definition
// in repository, create cache table as needed
df.createFeedDefinition();
```

12.2.3.3 Adding Input Parameter Values to the Feed

The data feeder only downloads content which has a specified input parameter. Input parameter values can be set either implicitly or programmatically. Input values can be added implicitly by adding an alert topic subscription. The following code illustrates how to add an alert programmatically:

```
// Look up existing data feeder
DataFeeder df = ModelServices.getInstance().lookupDataFeeder("stock_yahoo");
// Want to add input params for ORCL
Map params = new Hashtable();
params.put("sym", "ORCL");
df.setData(params);
```

12.2.3.4 Retrieving Downloaded Values

One primary use of the data feeder is to download cached data for use with regular synchronous services. Downloaded data can be accessed using the datafeeder method `getData()`. This method takes an argument as a map, which is a name-value mapping of the parameters which get values. The following code example illustrates how you can retrieve current price and change given a stock symbol:

```
ModelServices ms = MetaLocator.getInstance().getModelServices();
DataFeeder df = ms.lookupDataFeeder("stock_yahoo");
Map params = new Hashtable();
```

```
params.put("sym", "ORCL");
Map values = df.getData(params);
Iterator i = values.keys();
while(i.hasMore()) {
String key = (String)i.next();
String val = (String)values.get(key);
System.out.println(key + " = " + val);
}
```

Running this code we while get the following output:

```
sym = ORCL
price = 18.75
change = 0.5
```

12.2.3.5 Starting the Data Feeder Process

System managers start a data feeder process. Oracle9*i*AS Wireless engine. Like other processes, the system manager must set up a process of the datafeeder in order to run it. For more information, see the *Oracle9iAS Wireless Getting Started and System Guide*.

Note: The data feeder only downloads content where it has an input parameter value specified.

12.2.3.6 Feed Parameter External Names

The external name is the name used when retrieving content from a content provider. This mechanism is intended for cases where the external representation of the parameter name changes after the feed has been built, such as when one changes to another content provider. The external name is optional; if it is not specified, then the internal name is used.

You specify a caption to use for the input parameter. This is for documentation purposes only.

There are cases where an input parameter has been defined, but is not relevant when retrieving content. If the special constant __NONE__ is used for the external parameter name that input parameter will be ignored when constructing the download URL or POST request.

12.2.3.7 Feed Scheduling

By default, feeds run continuously when started. Each feed has an associated update policy, which can be used to fine-tune the running of the feed, such as the time of day to start and stop the feed, the days which to run and the interval between feed runs.

The following code sets the update policy of the example data feeder to run on weekdays between 9 am and 5 pm.

```
ModelServices ms = MetaLocator.getInstance().getModelServices();
DataFeeder    df = ms.lookupDataFeeder("stock_yahoo");
UpdatePolicy  up = df.getUpdatePolicy();
up.setStartTime(9,0,0);
up.setEndTime(17,0,0);
up.setUpdateDays(UPDATE_WORKDAYS);
// Set update interval to 300 seconds, i.e. update every
// 5 minutes
up.setUpdateInterval(300);
```

12.2.3.8 XML Data Feeds

When accessing datafeeds with XML content, you must specify a XSLT stylesheet that will transform the input XML to a common XML format.

The common XML format consists of a feed result (`<omfeed_result>`), which has a number (zero or more) of datarows (`<omfeed_datarow>`), each one consists of one or more named datacolumns (`<omfeed_datacolumn>`). The name of the data column is matched with the parameters defined for the feed. Each output parameter should have a corresponding data column. This code sample illustrates the output of a stock feed:

```
<?xml version="1.0"?>
<market-data>
<quote-set>
<quote symbol="ORCL" name="ORACLE CORPORATION" type="stock"
exchange-code="NASDAQ" last="32.000000" close="28.562500" close-flag="closed"
change="3.4375" percent-change="12.04%" volume="56362800" open="30.0"
high="32.4375" low="29.9375" bid="32.0" ask="32.0625" bid-size="36"
ask-size="90" high-52-week="46.468998" low-52-week="15.438"
shares-outstanding="5629833" pe-ratio="29.299999" volatlity="16.150000"
yield="0.000000" earnings-per-share="1.092000" status="ok"/>
</quote-set>
</market-data>
```

The stylesheet for transforming this result would then look like this:

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

```
<xsl:template match="quote-set">
  <omfeed_result>
    <xsl:for-each select="quote">
      <omfeed_datarow>
          <omfeed_datacolumn>
            <xsl:attribute name="name">sym</xsl:attribute>
            <xsl:value-of select="@symbol"/>
          </omfeed_datacolumn>
          <omfeed_datacolumn>
            <xsl:attribute name="name">price</xsl:attribute>
             <xsl:value-of select="@last"/>
          </omfeed_datacolumn>
          <omfeed_datacolumn>
            <xsl:attribute name="name">change</xsl:attribute>
            <xsl:value-of select="@change"/>
          </omfeed_datacolumn>
      </omfeed_datarow>
    </xsl:for-each>
  </omfeed_result>
</xsl:template>
```

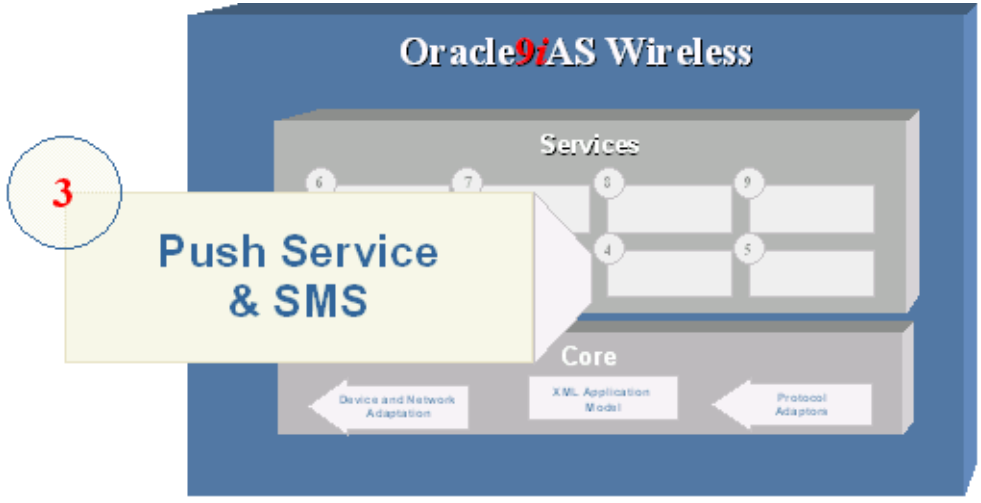
13

Push Service and SMS

This document describes Push and SMS Services architecture, and explains how to use these Services to create and deploy mobile applications. Each section of this document presents a different topic. These sections include:

- Section 13.1, "Push Service and SMS Overview"
- Section 13.2, "Push Services API"
- Section 13.3, "Oracle9iAS Wireless Messaging System"
- Section 13.4, "Oracle9iAS Wireless Pre-built Drivers"

Figure 13–1 Push Service and SMS



13.1 Push Service and SMS Overview

Push services are a key component that enhance your mobile application features by supporting alerts to your mobile users. Oracle9*i*AS Wireless Push services provide a highly scalable mechanism to deliver messages to all mobile devices. The messages are delivered to the mobile devices in a protocol that is native to the device, for example via SMS to a mobile phone, as an email to a 2-way pager, as an audio message to regular phone or as a fax to a fax machine.

Push Services in Oracle9*i*AS Wireless are implemented as a Web Service (WSDL) and use SOAP over HTTP. The SOAP service allows applications to invoke remote object methods over HTTP protocol. This enables applications to invoke push service from anywhere on the Internet and using any programming model. Oracle9*i*AS Wireless Push services allow applications to specify both the message and the recipient(s) of the message. The application communicates to the Push service in Oracle9*i*AS Wireless using SOAP and HTTP. Oracle9*i*AS Wireless receives the message and delivers the messages to mobile devices using appropriate protocols such as SMS, Email, Voice, and others.

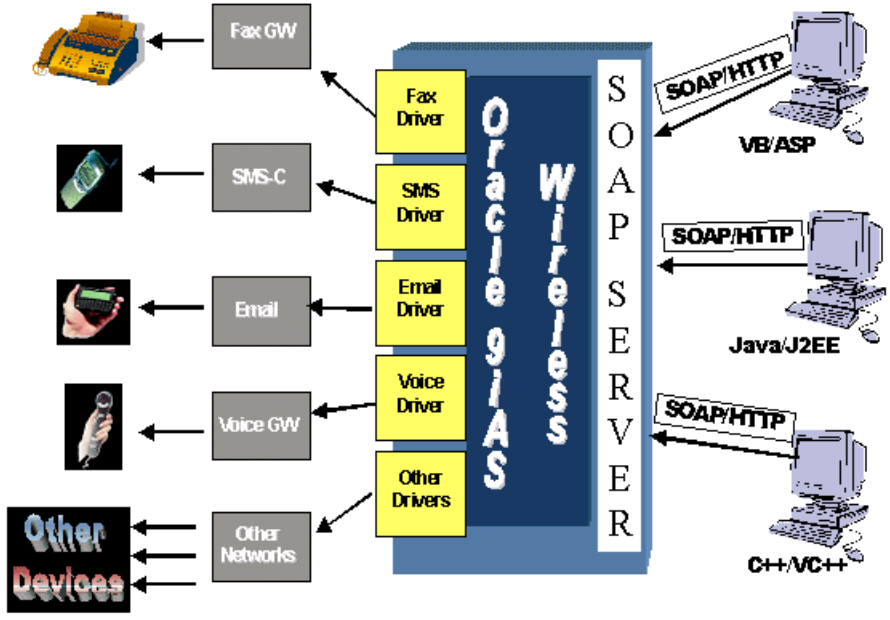
Figure 13–2 Push Services

Push Services in Oracle9*i*AS Wireless is a scalable platform that can handle large volumes of messages to many devices. The Push Service is based on extensible

architecture and design that can be extended to support a variety of devices and push protocols. Push SOAP messages are handled by a Messaging subsystem on the Oracle9*i*AS Wireless server. The Messaging subsystem supports a driver-based architecture. The drivers are components in the wireless messaging system that handle all device-specific or communication protocol specific routines. The Messaging subsystem, based on the device address and transport type (SMS, Voice, Email), dispatches the message to the appropriate transport / protocol driver implementation. The driver interface delivers the message to the device in the native device protocol. The Messaging subsystem can support multiple drivers in a single instance and can support multiple drivers.

Message drivers in Oracle9*i*AS Wireless are *plugable* modules that implement device-specific or communication protocol-specific handling routines. The Oracle9*i*AS Wireless server ships with pre-built drivers that support communication protocols such as SMS (SMPP and UCP), Voice, Email and Fax.

Figure 13–3 Message Drivers



Oracle9*i*AS Wireless server ships with a special driver implementation that allows your wireless instance (or install) to act as a client to another Oracle9*i*AS Wireless

installation or any service that respects the Web Service Interface defined by Oracle9*i*AS Wireless. This special driver uses the SOAP interface (as the PUSH APIs) to send the messages. In the product by default, this driver is configured to act as Push Client to an Oracle9*i*AS Wireless instance hosted, on the Internet, by Oracle. Instance administrators can change this default setting to point to any server that respects the PUSH WSDL interface defined by Oracle9*i*AS Wireless.

13.2 Push Services API

Oracle9*i*AS Wireless Push service is deployed as a web (WSDL) service using SOAP with HTTP as the transport layer. WSDL (Web Services Definition language) is a standard XML interface that defines a Web Service application. With clearly defined WSDL, developers can build applications in any programming language such as Java and VB, and communicates with the Oracle9*i*AS Wireless messaging interface over the Internet. Developer can use any WSDL toolkit to quickly implement a Push application and send messages to mobile devices using any Oracle9*i*AS Wireless instance on the Internet.

Oracle9*i*AS Wireless supports a simple Push API, in Java, that abstracts any protocol-specific (SOAP) implementation from the application Java code. The Java Push API is an interface in the Java programming language. It is the preferred API for application developers who need a clear and simple interface to deliver messages. The Java Push API uses SOAP over HTTP to communicate to the Oracle9*i*AS Wireless server instance.

Note: If you run your application, (hence the Java Push API) within the Oracle9*i*AS Wireless server Java VM, the Java Push API will not use SOAP, and instead will use in process communication to handle message delivery.

The Push API supports a uniform interface for the delivery of messages to any kind of devices such as SMS, Voice, Email, and Fax. The API allows applications to specify multiple recipients for a single message using only one delivery request. Further, the message destination addresses can have devices using different communications channels, for example, a single message delivery request application can send messages to Email as well as fax machines. Applications can make one delivery request and send the messages to a list of users with SMS devices, Email clients and Voice devices

Oracle9*i*AS Wireless supports different types of contents for delivery. A message can simply consist of only text characters, or can be as complex as a multipart

message. Message types are identified based on the MIME, hence delivering documents such as Microsoft Word or Adobe PDF is possible if the target device supports the message MIME type.

13.2.1 Building a Push Application

Oracle9*i*AS Wireless can support Push content of different MIME Types such as Microsoft Word documents or Ringtones. A message can consist of text only or can be a complex multipart message. Oracle9*i*AS Wireless identifies the message types based on the MIME, hence delivering document such as MS-Word or PDF is possible if the target device supports the message MIME type. Oracle9*i*AS Wireless provides Java Push API that support a text only message (PushLite) and an advanced Push API that supports messages of any MIME type (Push).

Following are the two interfaces to send a Push Message:

- Push Lite
- Push

13.2.1.1 PushLite

`oracle.panama.messaging.push.PushLite`

PushLite provides text messaging abilities. It is lightweight and very easy to use.

```
public String[] send(String[] senders, String[] recipients, String message)
```

Sends out a text message (without subject) to multiple recipients of multiple transport types. Encoding of the message is "text/plain". This method provides the easiest way to send out text messages. Other overloaded `send()` methods can be used to set subject, reply to, content type encoding (MIME type) and associated key parameters.

senders - an array of senders' addresses. A sender's address has a transport type and address, separated by a colon (:). One sender per transport type. Latest sender of the same transport type will override earlier senders of the same transport type in the array.

```
Example 1: "Email:myemail@company.com"
Example 2: "SMS:16505551234"
```

Valid transport types are defined in

`oracle.panama.messaging.common.TransportType`

recipients - recipients' addresses (Oracle9*i*AS Wireless email address or phone number)

Format of recipient addresses:

```
<transport>:<recipient address 1>[,recipient address 2 ...]
Example: SMS:1-650-5551234,1-408-3456789
```

Recipients of the same transport may be separated into multiple lines, But, these lines may not be separated by recipients of other transport type lines. An exception will be thrown if it has been detected.

```
Example 1 -- OK:
"Email:john@company.com,mary@company.com"
"Email:bob@company.com"
"SMS:1-123-45678"

Example 2 -- ERROR: second email recipients line (bob@company.com) is cut
off by SMS recipients
"Email:john@company.com,mary@company.com"
"SMS:1-123-45678"
"Email:bob@company.com"
```

message - body of message.

```
public String[] send(String[] senders,    String[] replyTOs,  String[]
recipients,  String[] associatedKeys,
           String subject,  String message,  String encoding)
```

sends out a text message with subject, reply to addresses, associated keys and content type encoding.

senders - senders' addresses (same as described above).

replyTOs - an array of alternate reply to addresses or phone numbers (optional). Use null if no reply to address. The format is the same as senders

recipients'' addresses (Same as above)

associatedKeys - an array of text strings that may be used by client applications to do message tracking. One key per recipient. The length of each key could be up to 64 bytes. The order of the keys are the same as the order of recipients. This field is optional, if no associated key is used, use null.

subject - subject of message (optional).

message - body of message.

encoding - MIME type with optional charset encoding of message. For example: "text / plain", "text / plain; charset=us-ascii" and "text / html"

```
public String getStatus(String messageID)
```

Get current status of one message ID.

Returns: a text status string

```
public String[] getStatus(String[] messageIDs)
```

Get current status of a set of message IDs.

Returns: an array of text status strings.

13.2.1.2 Push

```
oracle.panama.messaging.push.Push
public WorkOrder[] send(Packet pkt)
```

Send out a message packet.

pkt - The message packet to be delivered. Packet class will be discussed shortly.

Returns: a set of WorkOrders will be returned after the Push server accepts the request. One WorkOrder for each instance of recipient's address.

```
public Status getStatus(WorkOrder workOrder)
```

Get current status of a work order. One work order has one address and the message ID of that address.

```
public Status[] getStatus(WorkOrder[] workOrders)
```

Get current status of a set of work orders.

```
oracle.panama.messaging.push.Packet
```

Packet class represents a generic message in the real world. (For example: email) It may have a subject, one body or a set of message bodies (multipart). The same message may be delivered to multiple recipients of multiple transport types (delivery types).

For example: the same message can be delivered to 2 email recipients, 3 SMS recipients and 4 fax machines in the same packet.

Every transport type may have a sender, an alternate reply to address and a group of recipients. The packet could have a set of optional delivery instructions, such as priority, registered, etc.

To accomplish this, first construct an empty Packet instance. Then set message, message info, sender, reply to and recipients of the packet. Please see sample code below for more details.

The Push API provides methods to set the properties of a message and dispatch to the Oracle9*i*AS Wireless instance. For a detailed description of the API interfaces, refer to the Oracle9*i*AS Wireless Push Javadoc (oracle.panama.messaging.push). To send a push message you will need to provide the following:

- Oracle9*i*AS Wireless server where the Push Web Service is running. Include the username / password and the HTTP proxy required to access the remote Oracle9*i*AS Wireless Web Service (unless the application will be running in an Oracle9*i*AS Wireless VM).
- Actual message to be sent and the content (MIME) type of the message.
- Address of the target devices such as Email address, Phone number etc.
- How the message must be delivered, for example, as audio message over voice, SMS message over voice or as email message.
- Address of the sender of this message. The API allows you to set the sender address on a per transport type basis, such as one sender address for SMS, one for Email and one for voice.

To compile and run the Java Push API you will need JDK 1.3 for your platform and Oracle9*i*AS Wireless Push Java libraries.

13.2.1.3 Example: Send a message to multiple recipients

Use PushLite to send out message to two email recipients and 1 SMS phone.

```
     // 2 email and 1 SMS recipients
 String recipients[] = new String[2];
 recipients[0] = new String(TransportType.EMAIL + ":"
+"john@company.com,mary@company.com");
 recipients[1] = new String(TransportType.SMS + ":" +"1-333-5551234");
  // one email sender and one SMS sender
 String senders[] = new String[2];
 senders[0] = TransportType.EMAIL + ":" + "sender@company.com";
 senders[1] = TransportType.SMS + ":" + "1-222-1234567";
 String messageString = "Hello World!";   // message body
// set the gateway URL to null to use local Push Server.
```

```
// Local Push server is running in the same JVM of Push client. No SOAP is used.
String gatewayURL = "http://messenger.oracle.com/push/webservices";
 // create a PushLite client instance
 PushLite pushLite = null;
 try{
// In fact, Oracle's hosted Push Server does not require an account
// if you have not signed up, you can use "" as both username and password
pushLite = new PushLite(gatewayURL,"user name","password");
}
 catch(PushException e)
  { e.printStackTrace();   }

// Set proxy if Push client machine is inside firewall and Push Web Services is
outside firewall.
// Let's assume the proxy server is: www-proxy.company.com:80
 pushLite.setProxy("www-proxy.company.com", 80);

// get supported transport types on Push Web Services
String supportedTransportTypes[] = pushLite.getSupportedTransports( );

// send out a text message
  String wo[] = null;
  try
  {
    wo = pushLite.send( senders, recipients, messageString);
  }
  catch(PushException e)
  {
    System.out.println("**** PushException caught ");
    e.printStackTrace();
  }
  if(wo != null)
  {
    for(int i=0;i< wo.length;i++)
      System.out.println(wo[i]);
  }
// query delivery statuses
  String status [] = null;
  try{
    pushLite.getStatus(wo);
  } catch(PushException e) { e.printStackTrace(); }
  if(status != null)
  {
    for(int i=0;i< status.length;i++)
      System.out.println("[" + status + "]");
```

```
  }
```

13.2.1.4 Example: Sending an Oracle9*i*AS Wireless XML Message using PushLite

```
// In order to send out a Oracle9iAS Wireless XML message, we need to specify
oracle.panama.messaging.ContentTypes.MOBILE_XML as encoding type when calling
send( )

// String subject = "message subject";

// send out an Oracle9iAS Wireless XML text message
  String wo[] = null;
  try
  {
// no reply to addresses, no associated keys in this example
    wo = pushLite.send( senders, null, recipients, null, subject,
xmlMessageString, oracle.panama.messaging.ContentTypes.MOBILE_XML);

  }
  catch(PushException e) {  ...}
```

13.2.1.5 Example: OTA: Sending a Ringtone to two cell phones

```
import oracle.panama.messaging.common.*;
import oracle.panama.messaging.push.*;

        // SMS recipients
        AddressData smsRecipients[] = new AddressData[2];
        smsRecipients[0] = new PhoneAddressData("1-333-5551234");
        smsRecipients[1] = new PhoneAddressData("1-444-5551234");
        // Packet object
        Packet pkt = new Packet();

       AddressData smsSender = new PhoneAddressData("1-222-1234567");

        pkt.setFrom(TransportType.SMS, smsSender);

        pkt.addRecipients(TransportType.SMS, smsRecipients);

        Message msg = new Message();

      // Ring tone message.
       msg.setContentType(RingTone.MIME);
       msg.setSubject("ring tone");
```

```
       RingTone ringtone = new RingTone();
       ringtone.setRingToneEncoding(RingTone.RINGTONE_ENC_OTA_ASCII);
       ringtone.setPhoneModel("Nokia 6210");

ringtone.setRingTone("024A3A51D195CDD008001B2055059061056055855054854082084990000
");
       msg.setContent(ringtone);

       pkt.setMessage(msg);

        String gatewayURL = "http://messenger.oracle.com/push/webservices";

     // create a push client instance
        Push push = null;
        try{
          push = new Push(gatewayURL,"user name","password");
        }
        catch(PushException e) { e.printStackTrace();}

        WorkOrder wo[] = null;
        try
        {
      // send message packet to the server
                wo = push.send(pkt);
        }
        catch(PushException e)
        {
                System.out.println("**** PushException caught ");
                e.printStackTrace();
        }

        if(wo != null)
        {
                for(int i=0;i< wo.length;i++)
                System.out.println(wo[i]);

        // get sending statuses
                Status status[] = null;
                try {
                  status = push.getStatus(wo);
                } catch(PushException e) { e.printStackTrace(); }

                if(status != null)
                {
                for(int i=0;i< status.length;i++)
```

```
                                System.out.println(status[i]);
                }
        }
```

13.2.1.6 Using Push API - WSDL

The Push Service is deployed as a Web Service on the Oracle9*i*AS Wireless server. The Web service uses SOAP over HTTP. To enable development of push applications on non Java environments, Oracle9*i*AS Wireless provides a WSDL (Web Services Definition Language) file. The WSDL can be used with any Web services development toolkit that supports your application programming environment and Model. We also ship a standalone package to facilitate development using the Push server; this package is named push_client.zip and is available on the product CD. The WSDL file is provided in the wsdl directory of push_client.zip. Before your development, please verify the Push Web Services location at the end of the wsdl file:

```
<service name= "PushServer">
  <port ...>
    <soap:address location= "push web services URL" />
  </port>
</service>
```

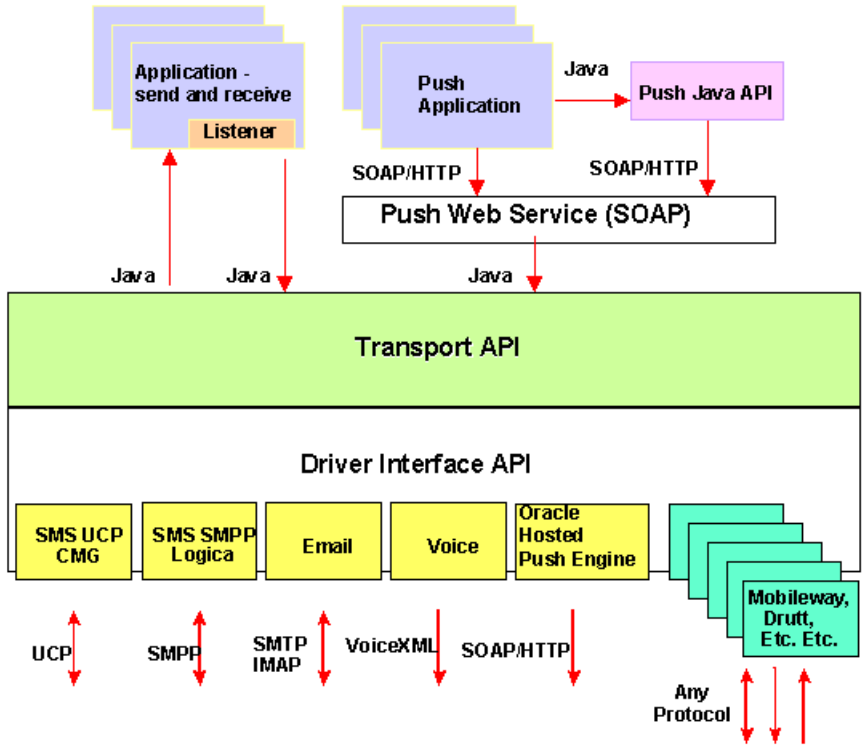
13.3 Oracle9*i*AS Wireless Messaging System

Oracle9*i*AS Wireless contains a Messaging subsystem that handles sending and receiving messages to and from devices, and all message routing functions. The driver, a component of this subsystem, implement the actual communication protocol stack while the Messaging system supports Transport API that provides the abstraction for applications to communicate with the Messaging subsystem. The Transport API (just as in the Push API) allows applications to send messages to mobile devices. In addition to sending messages the Transport API allows applications to receive messages from the client devices and also allows the applications to change message routing at runtime.

The Transport API application must be running on the same Java VM as the Oracle9*i*AS Wireless instance in order to use the Transport API (unlike the Push API which does not have this requirement). Oracle9*i*AS Wireless internal applications, such as the Async server and alert Engine, use the Transport API to deliver messages to the mobile devices.

Figure 13–4, "Messaging System Architecture" shows the Messaging subsystem architecture. To receive messages from the device, applications must implement a listener interface and register the listener instance with Messaging subsystem.

Figure 13–4 Messaging System Architecture



Transport APIs are independent of the underlying network protocols required to communicate with mobile devices. The underlying network protocols are implemented by the driver. The drivers implement device-specific and wireless communication specific messaging stacks. To deliver messages, the Messaging system uses the appropriate driver to deliver messages to the device. The Transport API supports operations that allow applications to send and receive messages from devices.

The Transport API provides Push applications with the necessary abstraction from the underlying wireless protocols and allows the Push application to be

device / network agnostic. The Transport API allows applications, if required, to control message routing with the available drivers.

The Messaging system supports multiple drivers for different wireless protocols and also allows multiple drivers that support a given network protocol. The Messaging system handles all message routing functions, techniques to boost performance, fault tolerance, and supports a highly scalable environment. The API allows developers to customize various functions of the transport system.

The drivers are *plugable* components in the Messaging system; they handle all network / protocol specific requirements. The Messaging system defines a Driver Interface that provides the necessary decoupling to achieve the required abstraction. The Driver API provides an extensible interface for network and protocol-specific drivers to be plugged into the Oracle9*i*AS Wireless Messaging system. A driver implementation wraps required network protocol routines for example, an Email driver can implement an IMAP / SMTP interface or use a MAPI interface. Drivers must implement the interface defined by the Messaging system.

13.3.1 Transport Runtime Processes

13.3.1.1 Push SOAP Web Service

The Push Web Service is a Servlet running as part of the VM for Oracle9*i*AS Wireless runtime, no special or independent process is started for it.

13.3.1.2 Messaging Servers

Messaging server processes are runtime containers for protocol drivers and provide the necessary environment for the drivers. The messaging server runs the driver instances and manages the life cycle of the driver instances. Oracle9*i*AS Wireless can run multiple messaging servers on different machines.

13.3.1.3 Driver and Driver Instance

Driver is the software that implements Oracle9*i*AS Wireless Transport Driver interface. A Driver instance is a runtime instantiation of a Driver that runs in a process.

Each driver identifies the type of protocol it handles such as Email or Fax. A single driver can handle more than one protocol. The protocol a driver handles, the implementation class and the drivers configuration parameter are specified using the Oracle9*i*AS Wireless Webtool.

The drivers are configured as a site level property for an Oracle9*i*AS Wireless installation and can be registered with one or more messaging servers. Such a configuration allows administrators to load balance and manage hardware resources based on the usage of individual drivers.

13.3.2 Configuration

13.3.2.1 Messaging Server

Using the Oracle9*i*AS Wireless Webtool a messaging server can be configured in the same way one adds, edits or deletes a server process. Also the Webtool enables you to configure performance parameters and the configuration of driver instances associated with the messaging server. See *Oracle9iAS Wireless Getting Started and System Guide* for more details.

13.3.2.2 Driver

The Oracle9*i*AS Wireless Webtool allows you to add, edit or delete a driver. Drivers are site-level properties in the Oracle9*i*AS Wireless installation. The configuration page allows you to provide details of a driver name, the implementation class and other driver-specific properties.

13.3.2.3 Driver Instances

Once a driver is made available on the site level, instances of the driver can be created and associated with individual messaging servers. You can have multiple instances of a driver running on different messaging servers, and these different instances can have distinct driver parameter settings. For example you could have one Email driver and create instances of the driver that point to different SMTP servers. The same is true for other protocols, such as multiple instances of the SMPP driver with different values for the various Telcos to which your server connects.

13.3.3 Transport API

This section details the Transport API, explaining the major constructs and functionality available to customize the Transport System.

Transport API is the client side messaging interface. Transport API is a rich set of APIs which can be used for both sending and receiving. In terms of sending, transport API provides some extra valued features, such as messaging routing and status tracking.

To receive messages, the application must register listening end points and a message callback listener to the transport system. An endpoint essentially is in the form of an address such as a phone number. It identifies to the transport system how message should be dispatched. When a message is received for a targeted address, it is dispatched to the listener associated with an endpoint with a matching address.

When a message delivery request is submitted, the transport system performs analysis of the recipients and routes the message to the appropriate protocol drivers for delivery.

13.3.3.1 Destination Analysis

A single message can be delivered to multiple recipients of different communication protocols. For example, one can send a meeting reminder to a few people using SMS, and some other people to their email addresses. So before routing messages to drivers, the transport must analyze and group recipients by their delivery category. Typically, the transport system starts its internal processing by analyzing all destinations and groups them accordingly.

13.3.3.2 Message Routing

To send a message, the transport system has to find a proper driver to do so. The process of finding a proper driver is called message routing. The transport system at a particular time may have many messaging servers and protocol drivers configured. Different driver instances may handle different categories of messages. For example, a driver may be able to send SMS messages only. Another one may be able to send email and fax messages only. Therefore, the transport system has to use a driver with SMS capability to send SMS messages, a driver with email capability to send email messages. Sometimes, there may be more than one driver that can handle the same category of messages. For example, there could be more than two SMS drivers. One talks to ATT's SMSC, the other talks to Cingular's SMSC. The transport system must use ATT's SMS driver to send SMS messages to ATT's devices, and use Cingular's SMS driver to send SMS messages to Cingular's devices. All these decisions are made by the transport based on two sets of information. The first set is the sending criteria specified by the application, such as delivery type, speed, cost, encoding and so on. Of these, the delivery type is required and can be specified in the class destination. The other set of information is provided by the set of available drivers. The properties of the drivers are configured by the administrator, such as driver speed, driver cost, encoding and delivery category. As mentioned earlier, routing finds the best matching driver. Some properties must match, for example, the delivery category; some of them just find the closest match, for example, cost and speed.

The transport uses the following info to do the routing:

- delivery category
- protocol
- carrier
- speed
- cost

Attribute encoding is not used in routing this release.

The transport will route a message to a driver with best match:

1. The delivery category, such as SMS or EMAIL.
2. The protocol, such as UCP or SMPP.
3. The carrier, such as Cingular or Telia.
4. If (speed_requested >= 0 and cost_requested >= 0), the minimum (driver_speed -speed_requested)**2 + (driver_cost - cost_requested)**2

   or

   if cost_requested < 0 the minimum abs(driver_speed - speed_requested)

   or

   if speed_requested < 0, the minimum abs(driver_cost - cost_requested)

   If more than one driver meet the above criteria, the transport chooses randomly one of them.

13.3.3.3 Providing hints to facilitate transport internal processing

Applications can provide hints that help speed up routing and destination analysis. For example, if you specify "Email" as the delivery category of all recipients, the transport will not have to look into each of the recipients to determine what they are.

In principal, the required parameter to deliver a message (the Messenger.send() methods) is Destination and Message. All others (SenderInfo, MessageInfo and DeviceInfo) are optional. When they are specified, they will be interpreted as hints that describe properties common to all recipients. For example, if a DeviceInfo is specified and the getDeliveryType() of this DeviceInfo instance returns DeliveryType.EMAIL.getName() then the transport will take it as a hint that all recipients are email addresses and no destination analysis will be performed.

13.3.3.4 Key interfaces/classes

The key interface is oracle.panama.messaging.transport.Messenger.

An instance of this interface will be returned via get method of oracle.panama.messaging.transport.MessengerController which in turn can be obtained through the TransportLocator class. This gives you access into the rest of the package to build your messaging applications.

Please refer to the javadoc for a complete reference of the APIs.

13.3.3.5 Hooks

Applications can install hooks that will be invoked during message sending and receiving depending on the type of the hook. All hooks are optional. Typically the hooks are passed all of the information the application specifies and can do what ever is appropriate. Hooks are useful in providing routing information, and perform other custom logic in some cases.

There are two main categories of hooks:

- **Named hooks**—only at most one hook for each kind and can be added only through webtool configuration.
- **General hook**—There are four kinds of general hook. They are: pre-send, post-send, pre-receive, post-receive. There can be none or multiple hooks for each kind and they can be added and removed either through the webtool or programmatically through methods available on the Messenger interface.

No default hook is provided for the product.

13.3.3.5.1 Named Hooks

DriverFinder—(interface oracle.panama.messaging.transport.DriverFinder). The expected semantics of this hook is to fill in the driver name for a delivery request.

CarrierFinder—(interface oracle.panama.messaging.transport.CarrierFinder). This hook is a named hook that can be configured through webtool. The expected semantics of this hook is to locate a carrier for a given device address. The carrier information is then used by the DriverFinder or the transport system to perform routing. It is generally called once per message. There can be only one hook of this kind.

GSMSmartMsgEncoder—(interface oracle.panama.messaging.transport.GSMSmartMsgEncoder). This hook is used to encode GSM smart messages.

FailOverHook—(interface oracle.panama.messaging.transport.FailOverHook). This hook is for future use.

13.3.3.5.2 General Hooks

PreSendingHook—(interface oracle.panama.messaging.transport.GeneralHook). This hook is called before sending any message.

PostSendingHook—(interface oracle.panama.messaging.transport.GeneralHook). This hook is called after sending any message.

PreReceivingHook—(interface oracle.panama.messaging.transport.GeneralHook). This hook is called before passing any received message to the listener.

PostReceivingHook—(interface oracle.panama.messaging.transport.GeneralHook). This hook is called after passing any received message to the listener.

13.3.4 OTA

Oracle Corporation has provided convenience classes to support Ringtone, Graphics and WAP Provisioning. To support other types of OTA such as calendar, see the documentation on the driver that provides such capability.

13.3.5 Sample programs

A sample program for sending.

```
/**
 * A simple transport client.
 *
 * @author jxiang
 */
public class SimpleClient {

  public static void main(String[] args) throws Exception {

    TransportLocator locator = TransportLocator.getInstance();
    MessagingController controller = locator.getMessagingController();
    Messenger messenger = controller.getMessenger();
    messenger.start();
    Destination dest = new Destination();
    dest.setAddress("1234");
    DeviceInfo di = new DeviceInfo();
    di.setDeliveryType(DeliveryType.SMS.getName());
    dest.setDeviceInfo(di);
```

```
    dest.SetDriver("SMSDriver");  // we know we are going to use SMSDriver. You
can also leave it blank.
    MessageInfo mi = new MessageInfo();
    dest.setMessageInfo(mi);
    SenderInfo si = new SenderInfo();
    Message msg = new Message();
    msg.setContentType("text/plain");
    msg.setSubject("subject");
    msg.setContent("body");
    String id = messenger.send(dest, si, msg, null);
    // try to get the sending status based on the message id.
    // general, you should query the status after some time,
    // allowing the transport to process.
    Status status = messenger.getStatus(id);
    messenger.stop();
  }
 }
A sample program for receiving.

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

 package oracle.panama.messaging.transport.test;

 import java.io.BufferedReader;
 import java.io.InputStreamReader;

 import oracle.panama.messaging.common.*;
 import oracle.panama.messaging.transport.*;
 import oracle.panama.model.DeliveryType;

 /**
  * A simple transport client.
  *
  * @author jxiang
  */
 public class SimpleClient {

   public static void main(String[] args) throws Exception {

     TransportLocator locator = TransportLocator.getInstance();
     MessagingController controller = locator.getMessagingController();
     Messenger messenger = controller.getMessenger();
     messenger.start();
```

```
    EndPoint point = new EndPoint("a@b.c", DeliveryType.EMAIL.getName());
    messenger.addEndPoint(point);
    point = new EndPoint("1-650-5061234", DeliveryType.SMS.getName());
    messenger.addEndPoint(point);
    messenger.setMessageListener(new ReceivingListener());
    messenger.start();
    System.out.print("Enter quit to quit: ");
    BufferedReader br = new BufferedReader(
      new InputStreamReader(System.in));
    while (true) {
      String buf = br.readLine();
      if (buf != null && buf.equals("quit")); {
        messenger.stop();
        break;
      }
    }
    System.exit(0);
  }
}


class ReceivingListener implements MessageListener {

  public int onMessage(String address, DeviceInfo info,
    String destination, Message message) {
    // process received message.
    String contentType = message.getContentType();
    if (contnetType.equals(Ringtone.MIME)) {
       // received a ring tone.
       // printed the attributes.
       Ringtone ringtone = (Ringtone)message.getContent();
       String enc = (String)ringtone.get(Ringtone.RINGTONE_ENCODING);
       // ............
    }
    return Listener.SUCCEED;
  }

}
```

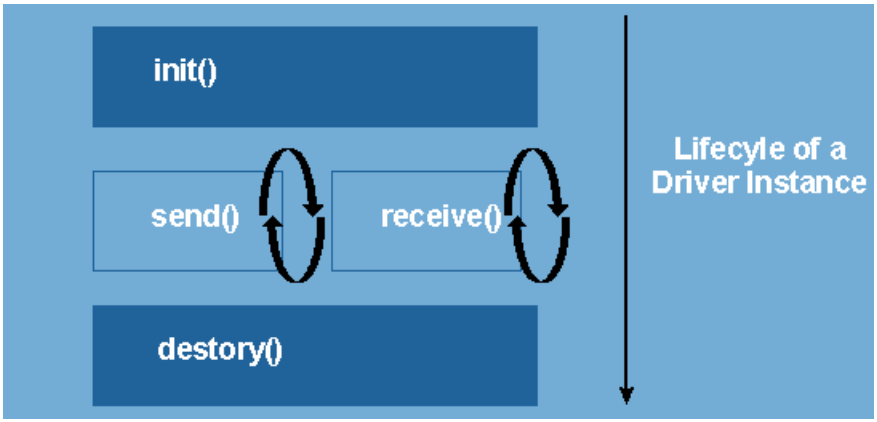
13.3.6 Driver Interface APIs

The driver interfaces are intended for the implementation of drivers for particular protocols. As explained above, drivers can be plugged into the transport system rather easily, extending network protocol support to the base product. A driver is

expected to be a very thin layer and handles only the protocol specific details. It should not deal with much of life cycle, load balancing or scalability issues. The transport system handles these issues.

The transport system initializes and destroys driver instances by respectively calling the init() and destroy() methods as specified in the Interface Driver. The transport system also handles load balancing and concurrence. A driver should just focus on interpreting the semantics of a particular protocol, leaving all others to the transport system.

Figure 13–5 Driver Lifecycle



A driver can be capable of only “sending”, or “receiving”, or both. To implement the “sending” semantics, a driver would just implement the send() methods as specified in the interface Driver. Receiving is a bit more complex in that the action to receive is driven by the transport. To implement receiving, a driver fills in the logic to receive in the receive() method specified by the Driver interface. The transport will continuously invoke the receive() method through out the life cycle of the driver instance.

However, the drivers should make every effort to be instance thread safe, or the usage must be clearly conveyed to system administrators so that proper configurations can be set to not thread the driver instance.

This following highlights the key classes and interfaces required for developing the SMS driver interface to work with the Oracle9*i*AS Wireless Platform.

> **Note:** All classes mentioned below assumes the package of oracle.panama.messaging.transport unless otherwise specified.

13.3.6.1 Class oracle.panama.messaging.transport.TransportLocator

The class TransportLocator defines interfaces that provides initial access to both the messaging interface and the driver interface. Two key methods defined for this class are:

- Method getDriverController() returns an instance for use of the driver interface; and,
- Method getMessagingController() returns a Controller instance for use of the messaging interface.

13.3.6.2 Interface oracle.panama.messaging.transport.Driver

This is the main interface for you to develop drivers for a particular protocol. You develop a driver by implementing the Driver interface. Your component is a qualified Oracle9*i*AS Wireless driver if it implements this interface.

13.3.6.2.1 The init() and destroy() methods These are the methods controlling the life cycle of the driver instance. The initialization properties passed to the init method are those specified through the Webtool configuration framework.

The init() method should return an initialization status, which can be one of:

```
Driver.FAILED, Driver.SEND, Driver.RECEIVE
Driver.SEND_RECEIVE.
```

Ensure the status returned is consistent with those configured through the webtool UI. If different, then the status returned here takes precedence.

13.3.6.2.2 The send() method Drivers implement this method to perform whatever is appropriate for their particular protocols to send out messages. The content to delivery is stored in the Message object passed onto the send() method, while the address parameter specifies one or more recipients to deliver the message to.

Further, the driver is expected to return a unique id for each message, or IDs one for each of the recipients. These ids will be used by the transport to query status of the delivery when necessary.

The driver must return a null message ID to make the transport retry. Exceptions thrown out of the send method are ignored, except for exceptions of type DriverException. If the send () method throws an exception of the type DriverException, the transport will not retry. If the code of the exception is marked fatal, the sending capability of this driver instance is revoked. If the exception is not marked fatal, the driver will still be used to send other messages.

13.3.6.2.3 The receive() method Drivers implement this method to perform whatever is appropriate for their particular protocols to receive messages. As mentioned above, the transport would drive the operation. Normally, the driver is expected to return from this method once a message is received. This way controlled is yielded back to the transport regularly so that the transport and decide the best step to take next.

The receive() method is called continuously by the transport. Hence it is preferable the receive() method blocks if it does not receive any messages. However it should not block indefinitely otherwise it will be considered a runaway operation and the thread that calls the receive will be terminated. The time elapse for runaway threads can be configured by setting "Maximum Execution Time per Request" under runtime configuration (default is 120 second). When a message is received, it should call the onMessage method of the Message Listener to submit the message (or, the onStatus callback on Status Listener if the message was a status report). The method can throw an exception of type DriverException and mark it fatal to ask the transport to stop calling the receive () method. The reason for this design is to simplify the logic and thread control of the driver.

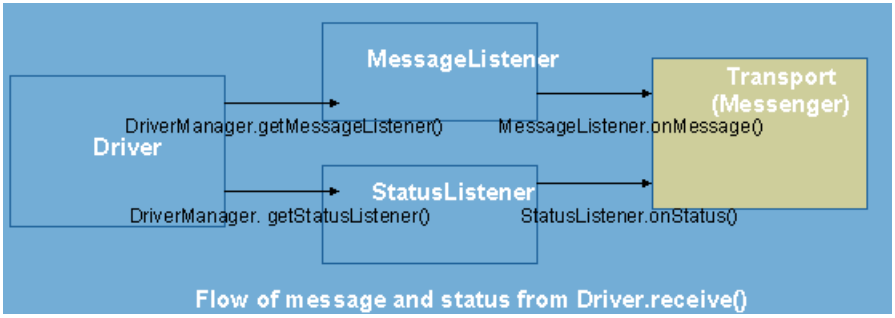
13.3.6.2.4 The getStatus() method The transport calls this method in an attempt to retrieve delivery status for a particular message.

13.3.6.2.5 The queryTracking() and queryNotifying() methods With some protocols, an active poll to the external service must be performed to check status of messages previously sent. These methods are called by the Transport to determine whether a getStatus() must be issued to retrieve status, or the driver would pass status back to Transport without such call (in this case, typically the driver calls onStatus() inside receive()).

13.3.6.3 Interface oracle.panama.messaging.transport.DriverManager

This provides entry point into other driver related utilities and interfaces such as Message Listener. You can get an instance of the DriverController by calling the getDriverController() method of the Locator.

Figure 13–6 Flow of Message and Status



13.3.6.3.1 The getMessageListener() and getStatusListener() methods These methods return the Transport callback instances for the receiving messages or status. You typically call the onMessage() or onStatus() methods within your implementation of the receive() method in the Driver interface to pass on messages and status to the Transport system respectively.

13.3.6.4 Interface oracle.panama.messaging.transport.GSMSmartMSGEncoder

If your implementation needs to handle UCP style smart message delivery, such as OTA WAP provisioning, ring-tone, graphics, you may find this interface useful.

We ship a default implementation of this interface, which can be located by getting the value for a property named “wireless.messaging.gsmsms.encoder.class”. The default implementation handles OTA WAP provisioning, ringtone, and graphics for Nokia and Ericsson handsets.

If you would like to extend the base capability you can do so by developing your own implementation by extending this interface. Once done, you should then configure the transport property "wireless.messaging.gsmsms.encoder.class” to have the value of the class of your implementation.

13.3.6.4.1 The encode() method This is the only method needed for the interface. The parameters indicate the type (ringtone, graphics), model (Nokia, Ericsson) and all the attributes relevant to the requested type.

You process the information and eventually return the encoded message in a form of GSMSmartMsg, which is essentially the fragments for the message and some specific smart message info.

It might happen that either the type, or model or other are of something your implementation does not support. In this case, you have two choices:

- Throw a Transport exception
  In this case, the smart messaging process terminates.
- Do not throw exception, just return null.
  In this case, our pre-built UCP driver will fall back to the default implementation of the GSMSmartMsgEncoder to see if it can handle the situation. Of course, this depends on whether a driver is developed with this semantics. However if you are focused on extending capabilities of the GSMSmartMsgEncoder you should follow this convention to allow maximum utilization of your development.

13.3.6.5 Interface oracle.panama.messaging.transport.MessageListener and StatusListener

You obtain instance of these interfaces by calling the appropriate methods in the DriverManager interface.

You use them typically within your implementation of the receive() method in the Driver interface to inform the availability of messages or status to the Transport system.

13.3.6.6 Class oracle.panama.messaging.common.Message

The message class is used to capture the content to be delivered or received. It is pretty comprehensive and has similar expressive power as email. It supports multi-part messages and allows mime types to be associated with the content. However, how to deal with the particular parts or MIME types is left for the implementation of the drivers.

13.3.6.7 Class oracle.panama.messaging.common.ContentTypes

This class is not a class only for drivers. It specifies a few content types (MIME types) in addition to the standard MIME types. As driver implementers, you might encounter these MIME types. How to deal with these MIME types is left to the individual driver, but it is critical that you are aware of them rather than failing when presented.

13.3.6.8 Properties of the driver

While adding a driver to the Oracle9*i*AS Wireless through webtool, a set of properties must be specified, as listed in the table below.

Table 13–1 Driver properties

Name	Description
Name	A discretionary name for the driver. This is a required property.
Class	The class that implements the driver. This is a required property.
Delivery Category	The supported categories of transport it supports, such as SMS, Email. This is a required properties.
Protocol	The particular protocol the driver transmits, such as SMPP or UCP. This is optional. Default is "*", meaning all of the possibilities.
Carrier	The carrier the driver can support, such as Cingular, Telia. This makes sense particularly in the SMS area and is optional. Default is "*", meaning all the possible carriers.
Speed	Speed of the driver. This can be used to better load balancing. This is optional, with possible value ranging from 0-10. Default is 0 (slowest).
Cost	Cost to use this driver. This can be used to improve load balancing. This is optional, with possible value ranging from 0-10. Default is 0 (most inexpensive).
Capabilities	Whether the driver can send, receive or both. This is optional defaulted to "send only".
Number of Message queues	This must be 1 (one) for this release.
Encoding, locale	Not used in this release.

13.3.6.9 Custom properties for a driver

When installing a driver, custom properties can be specified for the driver to function. For example, for an email driver to work, it might need to have a property for the imap hostname. The driver can be coded to expect a property of, say, name imap.hostname. When adding a driver through webtool, one can specify any number of property names. When creating the driver instances, the specific values of such a property can be provided. For example, out of the same driver code, one can install two email driver instances, each provided with imap hostnames to two distinct IMAP servers.

These set of custom properties will be passed into the driver instance when init() is called. In addition to the set of custom properties, some Oracle9*i*AS Wireless site-level properties are also passed implicitly, they are:

```
"wireless.log.directory";
"wireless.firewall.http.use.proxy";
"wireless.firewall.http.proxy.host";
"wireless.firewall.http.proxy.port";
"wireless.firewall.http.non.proxy.hosts";
"wireless.firewall.ftp.use.proxy";
"wireless.firewall.ftp.proxy.host";
"wireless.firewall.ftp.proxy.port";
"wireless.firewall.authentication.set";
"wireless.firewall.authentication.username";
"wireless.firewall.authentication.password";
```

13.3.6.10 Example: A Sample Driver

```
 // Copyright (c) 2001 Oracle Corporation. All rights reserved.
package oracle.panama.messaging.transport.driver.sample;

/**
 * A SimpleDriver class.
 * <P>
* @author Oracle Corporation
 */

import java.util.Properties;
import java.net.ServerSocket;
import java.net.Socket;
import java.io.BufferedReader;

import java.io.PrintStream;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.FileOutputStream;
import java.text.SimpleDateFormat;

import oracle.panama.messaging.transport.*;
import oracle.panama.messaging.common.*;
import oracle.panama.model.DeliveryType;
import oracle.panama.util.MessageCatalog;
import oracle.panama.core.admin.L;

/**
 * A Simple driver
 *
 * @author ashah
```

```
 */
public class SimpleDriver implements Driver {

  private String mCompanyName;
  private String mDeliveryType;
  private String mVersion;
  private PrintWriter log = null;

  /**
   * Initialize the driver.
   *
   * @param properties the driver's properties.
   * @return the initialization status.
   */
  public int init(Properties properties) {

    // get the locator instance and various listeners
    TransportLocator locator = TransportLocator.getInstance();
    DriverController manager = locator.getDriverController();
    mMessageListener = manager.getMessageListener();
    mStatusListener = manager.getStatusListener();

    // read properties
    mCompanyName = properties.getProperty("company-name");

    // delivery type is needed. Use SMS
    mDeliveryType = DeliveryType.SMS.getName();
    mVersion = "1.0";


    int status = Driver.FAILED;

    try {
      String logName = properties.getProperty("logfile");
      if (logName == null)
        logName = new String("SimpleDriver.log");
      log = new PrintWriter(new FileOutputStream(logName, true ) );
      } catch(Exception e) {
        e.printStackTrace();
      return status;
    }

    log ("initialized: " + new java.util.Date());
    mPort = Integer.parseInt(properties.getProperty("port"));
    mCounter = System.currentTimeMillis();
```

```
  mPrefix = "unknown::";
  mDelay = 20000;  // 20 seconds
  mMessage = new Message();
  mSemaphore = new Object();
  status = Driver.SEND_RECEIVE; // TODO - verify the return code

  mStatus = new Status();

  log ("init complete");
  return status;
}

/**
 * Destroy the driver.
 */
public void destroy() {
  try {
    log ("destroy");
    mServerSocket.close();
    mReader.close();
    mWriter.close();
    log ("destroy complete");
  }
  catch (Exception e) {
  }
}

/**
 * Get the version of the driver.
 *
 * @return the version of the driver.
 */
public String getVersion() {
  return mVersion;
}

/**
 * Get additional information of the listener.
 *
 * @return the information of the listener.
 */
public String getInfo() {
  return "Simple Driver";
}
```

```
/**
 * Send a message to a single address with this driver.
 *
 * @param address the address to send to.
 * @param encoding the encoding of the device.
 * @param tracking the tracking level.
 * @param expiration the expiration time.
 * @param reliability the reliability level.
 * @param fromAddr the from-address.
 * @param replyToAddr the reply-to-address.
 * @param message the message to send.
 * @return a unique message id, null if failed.
 */
public String send(String dtype, String address, int mode, String encoding,
  int tracking, int expiration, int reliability, String fromAddr,
  String replyToAddr, Message message) {

  log ("send: " + address + " => " + message.getContent());
  String id = null;
  try {
    id = mPrefix + getNextId();
    mWriter.println(id);
    mWriter.println(message.getContent());
    mWriter.flush();
  }
  catch (Exception e) {
    // not synchronized, it works for this toy.
    mWriter = null;
    mReader = null;
    id = null;
  }
  log ("sent id: " + id );
  return id;
}

/**
 * Send a message to a list of addresses with this driver.
 *
 * @param addresses the addresses to send to.
 * @param encoding the encoding of the device.
 * @param tracking the tracking level.
 * @param expiration the expiration time.
 * @param reliability the reliability level.
 * @param fromAddr the from-address.
 * @param replyToAddr the reply-to-address.
```

```
     * @param message the message to send.
     * @return a list of unique message ids, null if failed.
     */
    public String[] send(String dtype, String[] addresses, int[] modes, String
encoding,
      int tracking, int expiration, int reliability, String fromAddr,
      String replyToAddr, Message message) {
      String[] ids = null;
      log ("send: multiple => " + message.getContent());
      try {
        int count = addresses.length;
        ids = new String[count];
        String id = mPrefix + getNextId();
        ids[0] = id;
        mWriter.print(id);
        for (int i=1; i<count; i++) {
          id = mPrefix + getNextId();
          ids[i] = id;
          mWriter.print(',' + id);
        }
        mWriter.println();
        mWriter.println(message.getContent());
        mWriter.flush();
      }
      catch (Exception e) {
        // not synchronized, it works for this toy.
        mWriter = null;
        mReader = null;
        ids = null;
      }
      log ("sent multiple");
      return ids;
    }

    /**
     * Send a message to a list of addresses with this driver.
     *
     * @param dtypes the delivery types for all destinations
     * @param addresses the addresses to send to.
     * @param modes the delivery modes
     * @param encoding the encoding of the device.
     * @param tracking the tracking level.
     * @param expiration the expiration time.
     * @param reliability the reliability level.
     * @param fromAddr the from-address.
```

```
    * @param replyToAddr the reply-to-address.
    * @param message the message to send.
    * @return a list of unique message ids, null if failed.
    */
   public String[] send(String[] dtypes, String[] addresses, int[] modes, String
encoding,
     int tracking, int expiration, int reliability, String fromAddr,
     String replyToAddr, Message message) throws DriverException  {
     String[] ids = null;

     int count = addresses.length;
     log ("send: " + count + " recipients : " + message.getContent());
     ids = new String[count];

     for (int i=0; i<count; i++) {
       ids[i] = send(dtypes[i], addresses[i], modes[i], encoding, tracking,
expiration, reliability, fromAddr,
                     replyToAddr, message);
     }
     return ids;
   }


   /**
    * Get the sending status of a message.  The
    * status got by this call should be reported
    * the transport system via the driver listener
    * onStatus callback.
    *
    * @param mid the id of the message.
    */
   public void getStatus(String mid) {
   }

   /**
    * Get the sending statuses of a list of messages.
    * The statuses got by this call should be reported
    * the transport system via the driver listener
    * onStatus callback.
    *
    * @param mids the ids of these messages.
    */
   public void getStatus(String[] mids) {
   }
```

```
/**
 * Check if query is required to get the notification.
 *
 * @return true if required, false otherwise.
 */
public boolean queryNotifying() {
  return false;
}

/**
 * Check if query is required to track the
 * sending status.
 *
 * @return true if required, false otherwise.
 */
public boolean queryTracking() {
  return false;
}

/**
 * Receive a message/status.  If any message/status
 * is received, the driver should use the onMessage/
 * onStatus callbacks of the driver listener (got
 * via the controller) to report it to the transport
 * system.  This method should do something if the
 * initization status has the RECEIVE ability.
 */
public void receive() {
  log ("receive started");
  synchronized (mSemaphore) {
    try {
      if (mServerSocket == null) {
        try {
          mServerSocket = new ServerSocket(mPort);
          mServerSocket.setSoTimeout(mDelay);
        }
        catch (IOException ioe) {
          mServerSocket = null;
          mSocket = null;
          throw ioe;
        }
      }
      if (mSocket == null) {
        try {
          mSocket = mServerSocket.accept();
```

```
            mSocket.setSoTimeout(mDelay);
          }
          catch (IOException ioe) {
            mSocket = null;
            throw ioe;
          }
        }
        if (mReader == null) {
          mReader = new BufferedReader(
            new InputStreamReader(mSocket.getInputStream()));
          mWriter = new PrintStream(mSocket.getOutputStream());
        }
        String buf = mReader.readLine();
        log ("receive read: " + buf);
        if (buf.charAt(0) == '*') {
          String address = buf.substring(1);
          mMessage.setContent(mReader.readLine());
          DeviceInfo info = new DeviceInfo();
          info.setDeliveryType(mDeliveryType);
          info.setEncoding("7b");
          String from = "FROM-ME-TODO";
          mMessageListener.onMessage(from, info, address, mMessage);
          log ("message sent to message listener");
        }
        else {
          mStatus.setContent("received");
          mStatusListener.onStatus(buf.substring(1), mStatus);
          log ("status sent to status listener");
        }
      }
      catch (IOException ioe) {
        mReader = null;
        mWriter = null;
      }
    }
  }

  private synchronized long getNextId() {
    if (++mCounter < 1) mCounter = 1;
    return mCounter;
  }


    /**
   * write to message log
```

```
     *
     * @param message string
     */
    void log(String message) {
      if( log != null ) {
        synchronized( log ) {
          String currentTime = new SimpleDateFormat( "yyyy-MM-dd
HH:mm:ss").format( new java.util.Date() );
          log.println(currentTime + " " + message);
          log.flush();
        }
      }
    }

    private Socket mSocket;
    private Object mSemaphore;
    private ServerSocket mServerSocket;
    private MessageListener mMessageListener;
    private StatusListener mStatusListener;
    private BufferedReader mReader;
    private PrintStream mWriter;
    private Message mMessage;
    private Status mStatus;
    private String mPrefix;
    private long mCounter;
    private int mDelay;
    private int mPort;

  }
```

13.4 Oracle9*i*AS Wireless Pre-built Drivers

As explained before, drivers are plug-in components to Oracle9*i*AS Wireless that extends protocol-specific support of the system. Oracle9*i*AS Wireless ships with a few pre-built drivers that support major protocols that have been accepted as industry standards.

The pre-built drivers handle communication protocols such as SMS (SMPP and UCP), Email, Voice, Fax. The following section provides a brief overview of these pre-built drivers and the configuration properties required for these drivers. The pre-built drivers implement only a subset of the communication protocols specification that are sufficient to construct and deliver and receive a message. The pre-built Oracle9*i*AS Wireless drivers typically do not implement the complete specification of the communication protocols.

13.4.1 PushClient Driver

This driver uses a Hosted Push service and by default uses the service hosted by Oracle Corporation. This driver essentially acts like a Push client to an Oracle9*i*AS Wireless server hosted on the Internet and can be configured to point to any service that supports the Oracle9*i*AS Wireless Push Web Service. The PushClient driver uses a special protocol, SOAP over HTTP (the Oracle9*i*AS Wireless Push Web Service). As a matter of fact, the Oracle hosted push server does not require any account for access. If you have not signed up, you can use "" as both username and password. The URL for it is: http:/ / messenger.us.oracle.com / push / webservices

13.4.1.1 Class name

```
oracle.panama.messaging.transport.driver.push.PushDriver
```

13.4.1.2 Configuration

- messaginggatewayURL

  URL to the Hosted Push Web Service. This parameter is required.

  For example: http:/ / messenger.oracle.com / push / webservices

- username

  Name to use to authenticate against the Push Service. Push Web Services can determine whether username and password are required. If username is not required by Push Web Services, leave blank (empty string). "Bad username or password " will be returned from Push Web Services if either username does not exist or password of that username is not correct.

  For example: messaginguser

- password

  Password of the user specified in username field, to authenticate against the Push Service. Push Web Services can determine whether username and password are required. If username and password are not required, leave password blank (empty string). "Bad username or password " will be returned from Push Web Services if either username doesn't exist or password of that username is not correct.

  For example: 8Uh42g

- Content type it could handle:

It can handle all content types. The hosted Push Web Services to determine how to handle them.

- Drive runs on an HTTP connection. No explicit HTTP proxy setting is needed because the Push driver will take proxy setting of Oracle9*i*AS Wireless.

This driver handles sending only. It supports as many transport types as the Hosted Push Service. The actual types supported are dependent on which Hosted (Oracle9*i*AS Wireless Instance) service is running. The Oracle9*i*AS Wireless Service supports an API that describes the exact transports supported by an Instance.

13.4.2 Email Driver

13.4.2.1 Classname

```
oracle.panama.messaging.transport.driver.email.EmailDriver
```

The email driver supports SMTP in delivering messages, and either IMAP or POP3 in receiving messages. This driver can handle sending and receiving messages. Both IMAP or POP3 protocols are supported for receiving messages.

13.4.2.2 Configuration

- server.incoming.protocol

  This is the value for e-mail receiving protocol. The possible values are 'IMAP' and 'POP3'. Required only if e-mail receiving is supported on the driver instance.

- server.incoming.host

  The host name of the incoming mail server. Required only if e-mail receiving is supported on the driver instance.

- server.incoming.usernames

  The list of user names of the mail accounts the driver instance is polling from. Each name should be separated by ',' (comma), for example, 'foo,bar'. Required only if e-mail receiving is supported on the driver instance.

- server.incoming.passwords

  The list of passwords corresponding to the user names above. Each password is separated by ',' (comma) and should reside in the same position in the list as their corresponding username appears on the 'usernames' list. Required only if e-mail receiving is supported on the driver instance.

Example: 'foopwd,barpwd'

- server.incoming.emails

  The e-mail addresses corresponding to the user names above. Each e-mail address is separated by ',' (comma) and should reside in the same position in the list as their corresponding username appears on the 'usernames' list. Required only if e-mail receiving is supported on the driver instance.

  Example: 'foo@oracle.com,bar@oracle.com'.

- server.incoming.receivefolder

  The name of the folder the driver is polling messages from. The default value is 'INBOX'.

  server.incoming.checkmailfreq

  The frequency with which to retrieve messages from the mail server. The unit is in seconds and the default value is 3 seconds.

- server.incoming.autodelete

  This value indicates if the driver should mark the messages 'deleted' after they have been processed. The value could be 'true' or 'false' and the default value is 'false'. For POP3 protocol, the messages are always deleted right after they are processed.

- server.incoming.deletefreq

  The frequency to remove the deleted messages permanently. The unit is in seconds and the default value is 300 seconds. A negative value indicates the messages should not be expunged. For POP3 protocol, the message is expunged right after it is processed.

- server.outgoing.host

  The name of the SMTP server. Mandatory only if e-mail sending is required.

  Example: smtp05.oracle.com

- default.outgoing.from.address

  The default FROM address if one is not provided in the outgoing message.

Note: Only server.outgoing.host must be configured if the driver is going to be sending only.

The email driver supports SMTP in delivering messages, and either IMAP4 or POP3 in receiving messages. This driver can handle sending and receiving messages. Both IMAP4 or POP3 protocols are supported for receiving messages.

Note: Only server.outgoing.host must be configured if the driver is going to be sending only.

13.4.3 Voice Driver

The voice driver supports the Out Bound Call protocol supported by VoiceGenie. Currently, it has been tested only to work with a VoiceGenie gateway. This driver handles sending messages only. Although the driver can send messages only, it should be configured to have both sending and receiving capabilities for the driver to work.

This driver can handle content type text

ContentTypes.MOBILE_XML_URL

ContentTypes.MOBILE_XML_URL_REMOTE

ContentTypes.MOBILE_XML

ContentTypes.URL (only if the content type of the URL resource is one of above)

For other content types, the driver will throw a non-fatal driver exception.

13.4.3.1 Classname

`oracle.panama.messaging.transport.driver.voice.VoiceGenieDriver`

13.4.3.2 Configuration

- voicegenie.outbound.servlet.uri

  URL for the VoiceGenie Outbound Call Servlet. This is required with no default value. A sample looks like

  `http://rossini.us.oracle.com/servlet/com.voicegenie.outboundcallservlet.OutboundCallServlet`. The driver will use the site level proxy configuration in accessing this URL.

- voicegenie.outbound.servlet.username

  Username for the VoiceGenie Outbound Call.

- voicegenie.outbound.servlet.password

  Password for the VoiceGenie Outbound Call.

- voicegenie.outbound.servlet.dnis

  The phone number to be set as the caller. This is optional. The default value is 12345678.

- voicegenie.urlservice.path

  Servicepath to the prebuilt VoiceGenie service. This driver depends on a Oracle9*i*AS Wireless service based on the HTTP adapter. By default the Oracle9*i*AS Wireless installation has an HTTP Adapter service named "VoiceGenieURLService" to support this voice driver. This is a required parameter. There is no default value and one must look at a particular Oracle9*i*AS Wireless installation to obtain value for it. Here is a sample:

  ```
  foo.oracle.com:9000/ptg/rm?PAservicepath=/VoiceGenieURLService&PAsubmit=
  Submit
  ```

- voicegenie.driver.receive.host and voicegenie.driver.receive.port

  These are the IP host and port for the HTTP adapter to get sending content in Mobile XML format. The port should be used by this driver only. These are required.

  Note: This driver opens a port (as specified in voicegenie.driver.receive.port) and listens to HTTP traffic. It uses voicegenie.driver.receive.host and voicegeneie.driver.receive.port to compose a URL for Oracle9*i*AS Wireless HTTP adapter to contact the driver. This is required if you want to send a message that contains Oracle9*i*AS Wireless XML. Please make sure to provide the correct hostname and unique port number in order for the driver to function.

13.4.4 UCP Driver

UCP (Universal Communication Protocol) is one of the most popular GSM SMS protocols. The Oracle9*i*AS Wireless Server product ships with a pre-built implementation of the UCP driver as a driver that is capable of both sending and receiving.

This driver can handle content type text

- ContentTypes.RING_TONE

- ContentTypes.GRAPHICS
- ContentTypes.WAP_SETTINGS
- ContentTypes.URL (only if the content type of the related resource is one of above)

For other content types, the driver will throw a non-fatal driver exception.

13.4.4.1 Classname

```
oracle.panama.messaging.transport.driver.sms.UCPDriver
```

13.4.4.2 Configuration

- sms.account.id

  This is the account ID for the SMSSC. Generally, it should be the assigned short number by the operator. This is required.

- sms.account.password

  This is password assigned by the operator. It is used to open a session to the SMSC with UCP command 60.

- sms.ucptype

  Specifies which command to use in sending a message. The possible values are 01 and 51. The default value is 01, which means UCP command 01 is used to send a message.

- sms.server.host and sms.server.port.

  SMSC server information the driver uses to open a TCP/IP connection.

- sms.receiver.listener.port

  If the driver is in listening mode, this port is used by the SMSC to initialize the TCP/IP connection to pass received messages to the driver. If the sms.server.url is specified, this one will be used. Otherwise, it will be ignored.

- sms.server.url

  This is the URL for the driver to access the SMSC to send messages via HTTP connection. If specified, the sms.server.host and sms.server.port will be ignored. If it is not specified, then the sms.server.host and sms.server.port are required.

- sms.message.maxchunks

This is the maximum chunks for any single message allowed. Chunks after this number are ignored. The default is -1, which means no limit.

- sms.message.chunksize

This is the maximum size for each chunk in byte. The default is 150.

Notes: 1) If you have a direct TCP connection to the SMSC, the driver uses Command 60 to start a session with the SMSC. This allows the driver and the SMSC to communicate with one socket connection for sending, receiving and status. In this case the sms.server.url is not used.

2) If the connection you have to a SMSC is HTTP based then you should provide the value for sms.server.url and this is the URL the driver instance uses to send messages. Also sms.receiver.listener.port should be provided so that the driver instance opens binds to this port for incoming messages. In the HTTP connection case, sms.server.host and sms.server.port are not used.

3) sms.message.chunksize controls the size of each message in case the message total size is bigger than one SMS message. sms.message.maxchunks controls the maximum number of chunks allowed for each message. Those beyond that will be discarded.

13.4.5 SMPP Driver

SMPP (Short Message Peer to Peer) is one of the most popular GSM SMS protocols. Oracle9*i*AS Wireless Server product ships with a pre-built implementation the SMPP driver as a driver that's capable of both sending and receiving. The driver opens TCP connection to the SMSC as a transceiver, hence only one connection (initiated by the driver) is needed for all communication between the driver and the SMSC.

This driver can handle content type text ContentTypes.URL (only if the content type of the related resource is text). For other content types, the driver will throw a non-fatal driver exception.

13.4.5.1 Classname

```
oracle.panama.messaging.transport.driver.sms.SMPPDriver
```

13.4.5.2 Configuration

- sms.account.id

  This is the account ID for the SMSSC. Generally, it should be the assigned short number by the operator. This is required.

- sms.smpp.system.id, sms.smpp.system.type and sms.smpp.system.password

  These three attributes depend on your SMSC. Along with the short number assigned to you, the operator may also give you a system ID, type and password for you to log in to the SMSC.

- sms.server.host and sms.server.port

  SMSC server information the driver uses to open a TCP/IP connection.

- sms.message.maxchunks

  This is the maximum chunks for any single message allowed. Chunks after this number are ignored. The default is -1. A negative value means there is no limitation.

- sms.message.chunksize

  This is the maximum size for each chunk in bytes. The default is 150.

13.4.6 Fax Driver (RightFax)

This a Driver that support Fax message and supports RightFax (by Captaris) FAX protocol. The driver depends on the RightFax software package and the availability of a RightFax Fax server to deliver fax messages. This driver is capable of only sending messages.

This driver can handle any content type. It particularly recognize the following MIME types:

- text/xml
- application/msword
- application/msexcel
- application/msppt
- application/postscript
- application/octet-stream

In the case of the ContentTypes.URL, the driver will retrieve the content from the specified URL. The content and MIME type returned by this operation will become content and MIME type sent to the fax server.

13.4.6.1 Classname

```
oracle.panama.messaging.transport.driver.fax.RightFAXDriver
```

13.4.6.2 Configuration

- server.url

  URL to the RightFax server. This is required.

- server.account

  Account name to the RightFax server. This is required.

  All the below default attributes are optional. They are used to customize the cover sheet only.

  ```
  default.sender.name
  default.sender.corporation.
  default.sender.fax
  default.sender.phone
  default.sender.address
  default.sender.notes
  ```

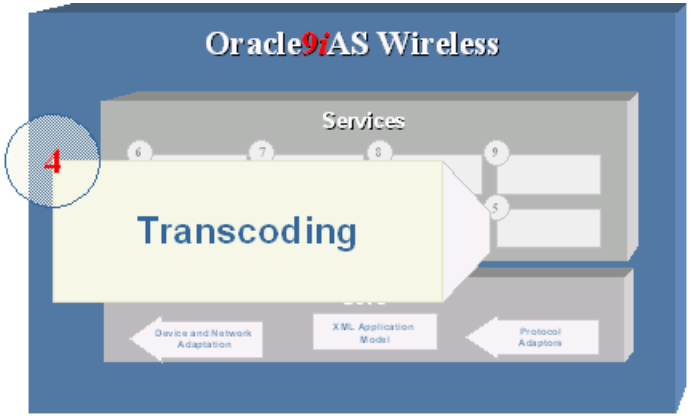
14

Transcoding

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

- Section 14.1, "Transcoding Overview"
- Section 14.2, "Web Content Adaptation"
- Section 14.3, "WML Translator"

Figure 14–1 Transcoding



14.1 Transcoding Overview

The majority of applications available on web render content in format specific to certain types of clients / devices. Transcoding services allow applications developed

for a particular device / markup language to be reformatted to formats suitable for the any web-enabled device.

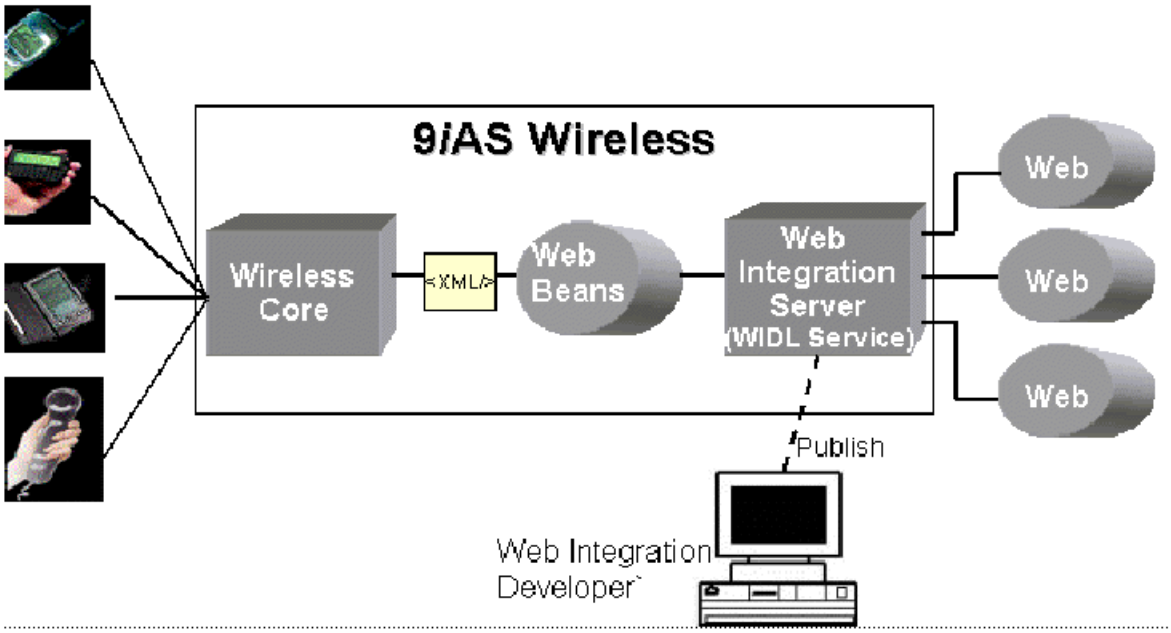
Transcoding services in Oracle9*i*AS Wireless supports a Web content reformatting service and a WML Translator service. The reformatting service allows Oracle9*i*AS Wireless applications to map and adapt any Web content to be reformatted for all web-enabled devices. The WML Translator service allows interpretation of content authored in WML and translates the content for access from all web enabled devices.

14.2 Web Content Adaptation

The Web Content adaptation service allows you to quickly extend your existing published and legacy web application to wireless web-enabled devices. Oracle9*i*AS Wireless services can connect any remote web resource, like HTML or XML document, and acquire content for reformatting. The content so acquired is adapted and mapped to data elements in wireless XML format and rendered to the Mobile devices.

Oracle9*i*AS Wireless supports WebIntegration Beans and Web Integration server that enables applications to map web content. The Web Integration Definition Language (WIDL), developed using Web Integration developer, defines the web content to be acquired by the Web Integration Server.

Figure 14–2 Web Content Adaptation and Reformatting.



14.2.1 WIDL Services

WIDL (Web Interface Definition Language) services allow you to acquire and extract content from any HTML documents or XML documents. WIDL service is an XML document that defines the web content that need to be acquires. WIDL services are individual units of programs that accept inputs from the application and returns outputs. The input and outputs elements are data structures with one or more elements. The inputs elements, if needed, are used by the WIDL service as inputs to acquire the content requested. The output elements represent the extracted data from the remote source. The elements within the input and output can contain complex data structures.

14.2.2 WebIntegration Beans

A WIDL service accepts inputs to execute a service and returns the extracted content as outputs. The inputs and outputs contain complex data structure elements. An application, to execute a WIDL service, has to connect to the Web Integration Server and manipulate the input and output elements.

WebIntegration Beans, a Java utility, provides the necessary abstraction and masks the complex nature of these input and output elements. WebIntegration Beans connects to the Web Integration server and executes the service. Also the beans provide software based round robin load balancing between different instances of Web Integration Server.

14.2.3 Using WebIntegration Beans

WebIntegration Beans supports the following classes

14.2.3.1 WebBeanContextDelegator

This object allows you to specify the service and the sub service of the Web Integration Service that needs to be invoked.

14.2.3.2 WebBeanDelegator

The WebBeanDelegator invokes the service with required service inputs and returns the outputs. The server to connect to is declared in the property file WebBeanProperty.properties. Both the inputs and outputs for the WebBeanDelegator are instances of HashMap (java.util.HashMap). The HashMap contains the collection of service input and output values. Depending on the service that values in the HashMap can be a string, HashMap or an array of HashMap.

14.2.3.3 Walkthrough: Creating an WIDL Using Web Integration Developer.

In this walkthrough, you create a WIDL Integration Wireless WIDL using the Developer tool. You would use the Web Integration Developer to map the elements of a sample Web page (http://finance.yahoo.com), and define input and output parameters. You will then publish this Service to the Web Integration Server.

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

14.2.3.4 Start the Web Integration Developer

1. Click Start on the Windows NT desktop and point to Programs.
2. From the Programs menu, select Oracle for Windows NT.
3. Select Wireless Edition, then select Web Integration Developer. The Web Integration Developer appears.

14.2.3.5 Open the Source Page

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

1. Select Open URL from the File menu or toolbar.
2. In the Open URL dialog box, type the following URL:
   http://finance.yahoo.com
3. Click OK. The Web Integration Developer retrieves the page, parses it, and adds it to the Document Browser (in the left frame). For this example, the Document Browser shows the following items:
   - The first item (http://finance.yahoo.com) identifies the open document.
   - The second item (the Document node) represents the contents of the HTML document: paragraphs, images, links, lists, and tables.
   - The third item (the FirstForm node) represents the form in this document. When you open a page that contains forms, the Web Integration Developer creates a form node for each form in the document.

14.2.3.6 Generate a WIDL File

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

1. Select FirstForm in the left frame.

   Note: Go to the Configure menu and set proxy to go through the firewall.

2. Select WIDL from the Generate menu.
3. Complete the New Service dialog as follows:

   In this Field... Type...
   Interface `StockInfo`
   Service `Yahoo_GetQuote`
4. Click OK.
5. In the Generate New WIDL for Service dialog, type `ORCL`.
6. Click the Submit button.

14.2.3.7 Edit the Input Binding of the WIDL File

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

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

14.2.3.8 Edit the Output Binding of the WIDL File

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

1. Select the Yahoo_GetQuoteOutput binding in the left frame.
2. In the right frame, scroll through the variable list until you reach the table variables.
3. Click the variable table10. Make sure the content of this variable is the stock price of ORCL. If not, click the variable whose content is ORCL's stock price.
4. On the Sample tab, select the cell that contains the current stock price.
5. Click the right mouse button and select Create New Variable From Selection from the pop-up menu.
6. Type `CurrentPrice` in the New Variable dialog and click OK. The `CurrentPrice` variable now appears in the variable list.
7. Delete all other variables from the variable list.
8. Save the WIDL file.

14.2.3.9 Test the WIDL File

To test the WIDL file in the Web Integration Developer:

1. Click Yahoo_GetQuote in the Services folder in the left frame.
2. Select Test Service from the Tools menu.

3. Type any valid stock symbol in the CoSymbol field.

14.2.3.10 Publish the WIDL Interface to the Web Integration Server

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

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

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

1. Select StockInfo in the left frame.
2. From the File menu, select Publishing, then Publish Interface.
3. In the Specify Server field in the Publish Interface dialog, type the name of the Web Integration Server to which you want to publish this interface. Specify the server name using the format:

   host_name:port
4. The Web Integration Server uses packages to organize services. You can click Update Packages to view a list of packages on the specified server, then add the service to a specific package. In this case, however, you can add the sample service to the Default package. Click OK.
5. If the User Name and Password dialog appears, enter a user name and password for the selected server. This user must have administrative privileges.Click OK.

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

14.2.3.11 Walkthrough: Developing an Oracle9*i*AS Wireless Service with Web Integration Service

This walkthrough continues our StockInfo WIDL Service (from previous WIDL walkthrough). We will create a JSP application that generates Oracle9*i*AS Wireless XML and uses the WebIntegration Beans to execute the StockInfo WIDL service. This JSP application will then be deployed as an Oracle9*i*AS Wireless HTTP Service and render to the Wireless Device.

14.2.3.12 Create the JSP Application

Create a JSP file, shown in the example. This JSP generates Oracle9*i*AS Wireless XML. The JSP looks for the "CoSymbol" as a request parameter, if Present executes the WIDL Service using the WebIntegration Beans. The response from the JSP contains the Price of the Stock requested and is embedded in Oracle9*i*AS Wireless XML format.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE SimpleResult PUBLIC "-//ORACLE//DTD SimpleResult 1.1//EN"
"http://xmlns.oracle.com/ias/dtds/SimpleResult_1_1_0.dtd">
<%@ page language="java" session="false" %>
<%@ page import="java.util.*" %>
<%@ page import="oracle.panama.tools.webbean.*" %>
<%
    String CoSymbol = request.getParameter ("CoSymbol");

    //CoSymbol is null, ask for a Symbol from the User
    if ((CoSymbol == null) || (CoSymbol.length() == 0)) {
%>
      <SimpleResult>
        <SimpleContainer>
          <SimpleForm target="StockQuote.jsp">
            <SimpleTitle>Stock Quotes</SimpleTitle>
            <SimpleFormItem name="CoSymbol" type="none" displaymode="text">
              <SimpleTitle>Enter Company Symbol</SimpleTitle>
            </SimpleFormItem>
          </SimpleForm>
        </SimpleContainer>
      </SimpleResult>
<%
    }
    else {

      //Set the Input to the User given Symbol
      HashMap inputs = new HashMap();
      inputs.put("CoSymbol",  CoSymbol);

      //Define the Service and ServiceContext
      //Set the Service to "StockInfo" and SubService to "Yahoo_GetQuote"
      WebBeanContextDelegator context = null;
      context = new WebBeanContextDelegator();
      context.setService("StockInfo");
      context.setSubService("Yahoo_GetQuote");

      //Connect to the Server and Invoke the Service
```

```
      WebBeanDelegator  webBean = null;
      webBean = new WebBeanDelegator();
      HashMap outputs = webBean.invokeWebService(context,inputs);

      String CurrentPrice = (String)outputs.get ("CurrentPrice");
%>
      <SimpleResult>
        <SimpleContainer>
          <SimpleText>
            <SimpleTextItem>
               Current Price of <%=CoSymbol%> is: <SimpleBreak></SimpleBreak>
               <%=CurrentPrice%>
            </SimpleTextItem>
            <SimpleAction type="primary" target="StockQuote.jsp" label="New"></S
impleAction>
          </SimpleText>
        </SimpleContainer>
      </SimpleResult>
<%
    }
%>
```

1. Deploy this JSP on your J2EE Web Application Server.
2. Confirm if you can access this JSP from your desktop browser. For example: Internet Explorer (5.0) would display this page as an XML file, while Netscape will just display the text in the file.

14.2.3.13 Creating a Oracle9*i*AS Wireless StockQuotes Service

You will now use the Oracle9*i*AS Wireless Webtool to create a StockQuotes Service. The StockQuotes is an Oracle9*i*AS Wireless service and will execute the StockQuote.jsp using the HTTPAdapter / HTTPMasterService. The Oracle9*i*AS Wireless XML returned by the JSP is the transformed to the device Markup Language

14.2.3.14 Creating an HTTPAdapter Service

1. Open the Webtool with your HTML Browser and Login
2. Navigate to the Content Manager tab on the Webtool.
3. Select “Add Service” link on the bottom of the Content Manager Page.
4. Complete the New Service page as follows and click
   Service Name: StockQuotes

Description:Mobile Stock Quotes
Select Visible Check Box
Select Personalizable Check box
Select Normal Service

5. Click Next
6. Select HttpMaster Service
7. Click Next. The screen for Setting the Init Parameters appears. Enter the following information into the fields.
   a. For the URL, enter the URL to your JSP application in the Value Column. It will be in the form of http://yourserviceport/path/.
   b. For the Replace URL enter true in the value column.
   c. For the Form Method enter GET in the value column.
8. Click Next and on the Next page click Submit to create the Stock Quotes Service.

14.2.3.15 Making Stock Quotes Service Available to a Group

You will Webtool to make the service you created available to a group of users:

1. Click the Content Manager tab on the Webtool
2. On the Content Manager Page, select the sub tab Groups.
3. Select the group Guests and click Assign Services.
4. From the list of Available services, on the bottom half of the page, select StockQuotes Service.
5. Click on AddToGroup button. This makes the service available to both guest and registered users of Oracle9*i*AS Wireless.

14.2.3.16 Testing Stock Quotes

You will now test the StockQuotes service using a browser and a phone simulator.

14.2.3.17 Testing the Service on a Browser

Log on to a browser to test the service. To access the URL:

1. From a Web browser, enter the following

   URL: http://9iASWEServer.domain/ptg/rm

2. Click StockQuotes Link
3. Enter a valid stock symbol and click submit.

   The StockQuotes retrieves and displays the current stock price.

14.2.3.18 Testing Stock Quotes on a Phone Simulator

You can use any phone simulator (for example, WML, HDML, CHTML) to test Stock Quotes. If you do not already have one, download and install a simulator. If you are working behind a firewall, you must configure the proxy server settings in the simulator before using it to access external sites.

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

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

   http://9iASWEServer.domain/ptg/rm
   This is the URL of the device portal for your Wireless Server installation.
3. Select the StockQuotes Link
4. Enter a valid stock ticker symbol (for example, ORCL) and click OK.

The Wireless server retrieves and displays the current price of the stock.

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

14.3 WML Translator

The WML translator service reformats WML documents/resources on the web to be available on all wireless web-enabled devices. The WML translator performs On-the-fly translation of remote WML resource into Oracle9*i*AS Wireless XML. The wireless XML is then transformed into appropriate device specific markup language.

This WML Translator enables these legacy WML applications to be integrated into mobile portals, deployed on Oracle9*i*AS Wireless, accessible by all web-enabled devices.

14.3.1 Deploying and Configuring WML Translator

The WML Translator is deployed as an Oracle9*i*AS Wireless module. The Translator service has the following service parameters.

- ORACLE_SERVICES_COMMERCE_TRANSLATOR_DEFAULT_CONNECTION

  This represents a fully qualified class name. This class encapsulates the connection to the remote WML resource. This class will be instantiated by the WML Translator service and is used to get content from a WML URL.

  The connection is defined by the Interface oracle.panama.module.commerce.translator.ConnectionIfc.

  This default connection implementation uses HTTP connection to retrieve the WML content and is implemented by the class oracle.panama.module.commerce.translator.WMLConnectionImpl

- ORACLE_SERVICES_COMMERCE_TRANSLATOR_HELPER_WML

  This is a fully qualified class name that implements the WML document translation Interface defined by oracle.panama.module.commerce.translator.WMLTransformImpl. This class will be instantiated by the WML Translator service and is used to transform WML document to wireless XML format.

  This default implementation uses the standard XSL Stylesheet and XSLT processor to perform the required transformation. The default implementation is provided by the class oracle.panama.module.commerce.translator.WMLConnectionImpl

- ORACLE_SERVICES_COMMERCE_TRANSLATOR_XSL_WML_FILENAME

  This is an URL to the location of the pointing to the location of the XSL stylesheet that is used to transform WML document to Oracle9*i*AS Wireless XML. In no value is specified, then the default transformation class uses a pre-built XSL stylesheet.

14.3.2 Using the WML Translator

The WML Translator is deployed as an Oracle9*i*AS Wireless Module service with module URL being omp://oracle/services/commerce/translator. To use the WML Translator applications can invoke the module, with the URL to the WML application as a parameter. The default parameter name is XLTORSITE, for e.g. to invoke to www.oraclemobile.com you can use the following URL in your

Oracle9*i*AS Wireless XML
omp:/ / oracle / services / commerce / translator?XLTORSITE=http%3A%2F%2Fwww.oraclemobile.com

15

Using Location Services

This chapter provides conceptual and usage information for developers of location-based applications. It contains the following major sections:

15.1 Introduction to Location Services

Developers of location-based applications need specialized services for:

- Geocoding: associating geographical coordinates with addresses
- Mapping: providing a graphical map for a point, set of points, route, or driving maneuver
- Routing: providing driving directions
- Business directories (Yellow Pages): listing businesses by region by either category or name
- Traffic: providing information about accidents, construction, and other incidents that affect traffic flow

Several companies provide these types of specialized content and applications. For example, some Web sites have categories for business directories, and some sites provide driving directions. Developers building mobile applications based on the Oracle9*i*AS Wireless framework can benefit from being able to use the specialized content and services. It is inefficient for each application to write custom interfaces to all the services that it wants to access.

The Oracle9*i*AS Wireless location application components are a set of APIs (application programming interfaces) for performing geocoding, providing driving directions, and looking up business directories. Service proxies are included that map existing important providers to the APIs, and additional providers are expected to be accommodated in the future.

Oracle9*i*AS Wireless application developers can take advantage of a uniform interface to access different service providers without having to make any changes to their applications. They can also use the infrastructure to prioritize services based on criteria such as quality, availability, or cost. Service providers also benefit from the fact that their contents and specialized functions are available "out-of-the-box" to all Oracle9*i*AS Wireless application developers.

This section introduces the location application components API, describes how to find the detailed javadoc-generated documentation and online examples, and explains conceptual and usage information that you must understand before using the components. It contains the following major subsections:

15.1.1 Getting Started

To get started using the Oracle9*i*AS Wireless location application components, follow these steps:

1. Read the conceptual and usage information in this document before using any example programs or creating any applications.
2. Go to the `sample` directory, which contains example files. Read the `Readme.txt` file in that directory; examine the supplied files, and use any that meet your needs.

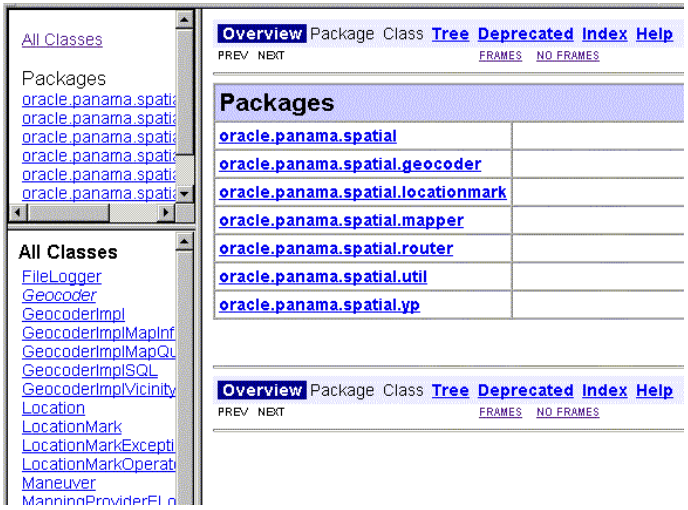
3. View the javadoc documentation, and refer to it for detailed reference information about packages and classes. To view the javadoc documentation, open the following file in a Web browser:

   iAS-Wireless-Home/wireless/doc/index.html

   where *ias-Wireless-home* is your Oracle9*i*AS Wireless home directory.

   Figure 15–1 shows part of the `index.html` display. Navigate to find detailed information about packages and classes.

Figure 15–1 Javadoc Documentation



15.1.2 Location Services

Location services are provided in the following major categories: geocoding, mapping, routing, business directory (Yellow Page), and traffic.

Other sections in this chapter describe how to specify and configure external providers for location services, and describe each type of service in greater detail.

15.1.2.1 SpatialManager Class

The `SpatialManager` Java class manages all these location services.

The `SpatialManager` class is defined as follows:

```
package oracle.panama.spatial;
import …;
public class SpatialManager
{
  public static synchronized Geocoder  getGeocoder() {…}
  public static synchronized Router  getRouter() {…}
  public static synchronized YPFinder  getYPFinder() {…}
  public static synchronized Mapper getMapper() {…}
  public static synchronized TrafficReporter getTrafficReporter() {…}
}
```

15.1.3 Service Providers

The actual core computation for location services is generally performed at an external provider. The external provider might be accessed over the Internet or other means of communication, or might be local. The Oracle9*i*AS Wireless Location Application Components API performs the communication and adaptation of results in a unified framework, so that users are generally not aware of which provider is supplying a particular service. In addition, the API minimizes the application developer's implementation effort and dependence on specific providers.

Access to an initial set of providers for most services is included. Some providers have full configuration information included, and some do not. (For providers that do not have configuration information, you usually receive the necessary information after you purchase the right to use their user name and password.)

You can provide access to additional providers by using the webtool. If a a new provider is added and if the provider does not use the same interface as an existing provider, a Java class must be created to translate between the provider's format and the Oracle9*i*AS Wireless location application components API. (This program is specified as the `ProviderImpl` attribute.) In addition, the implementing class file for the program must be added to the class path.

Using multiple providers for a service increases the probable reliability of the service. The API fails only either if all providers fail or if Web access is temporarily unavailable. Because providers are specified in preference list, the API automatically fails over when the preferred provider cannot perform the requested service, such as when any of the following occurs:

- The provider is temporarily inaccessible over the Web.
- The provider does not support the exact requested service.
- The request is incorrectly specified (such as a nonexistent address).

15.1.3.1 Provider Selection

Location services use a list of providers and support fail-over between them. The sequence in which providers are tried should ideally represent an order of preference. The preference ranking can be a simple ranking of providers, or it might be affected by region, time, performance, reliability, and cost.Whichever criteria are used, they are evaluated by a provider selection framework that determines provider order of preference.

The provider selection framework needs to be configured, as described in this section. If a service request is not satisfied by the framework, then either the provider selection framework implementation has been incorrectly configured or all providers have failed. You can find information about any problems or failures from the console log or the log file, as explained in Section 15.1.3.2.

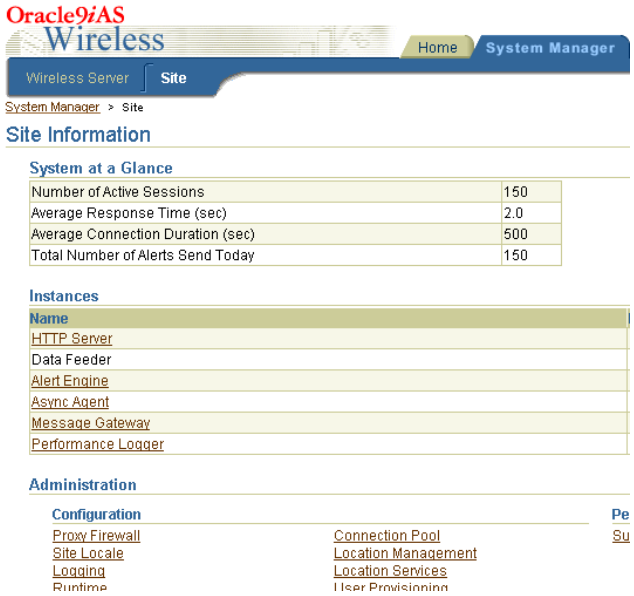
You must select a provider selection framework to be used. To select the framework, use the Oracle9*i*AS Wireless webtool and follow these steps:

1. If the System Manager page is not already displayed, click the **System Manager** tab.
2. Click the **Site** tab on the System Manager page. The Site Information page is displayed, the top portion of which is shown in Figure 15–2.

Figure 15–2 Site Information Page

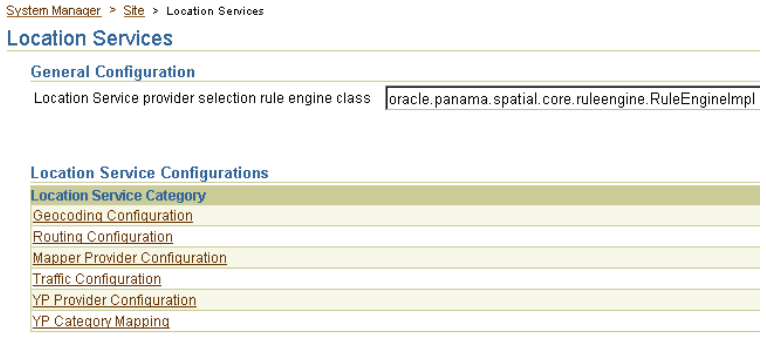
3. Scroll down to the Administration section and Configuration subsection. Figure 15–3 shows the portion that includes Location Services.

Figure 15–3 Administration Section of Site Information Page



4. Click **Location Services** in the Configuration subsection. The Location Services page is displayed, as shown in Figure 15–4.

Figure 15–4 Location Services Page



5. On the Location Services page under Basic Configuration, for **Provider Selector Class Name** enter a provider selection framework implementation.

Your choice of a provider selection framework implementation determines whether more or less complex rules can be used for provider selection. The following implementations are available:

- `oracle.panama.spatial.core.ruleengine.SimpleRuleEngineImpl`

  This simple implementation tries all providers until one succeeds. The sequence in which providers are tried is specified in the provider configuration list.

- `oracle.panama.spatial.core.ruleengine.RuleEngineImpl`

  This implementation can select providers based on whether or not they provide satisfactory coverage for a given country. Among all providers that provide satisfactory coverage for a given country, providers are tried based on the sequence in the provider configuration list.

  This implementation avoids time being wasted trying providers that do not provide coverage for a country or that provide unsatisfactory service (for example, if the cost is too high or the service quality is poor). However, this selection framework does require more configuration: lists of countries and country aliases need to be specified for each provider (although examples of such configurations are provided).

Other provider selection framework implementations can be added later.

15.1.3.1.1 Configuring Provider Information To configure the provider information, use the Oracle9*i*AS Wireless webtool and go to the same page as for provider selection, then select the appropriate type of service for configuration. Follow these steps:

1. If the System Manager page is not already displayed, click the **System Manager** tab.
2. Click the **Site** tab.
3. Scroll down the page to the Administration section and Configuration subsection.
4. Click **Location Services** in the Configuration section.
5. Select the appropriate type of service for configuration:
   - **Geocoding Configuration**
   - **Routing Configuration**
   - **Mapper Configuration**

- **Traffic Configuration**
- **YP Provider Configuration**

The provider information (described in Section 15.1.3.1.2) is very similar for all types of services (geocoding, mapping, routing, traffic, and YP).

For geocoding and perhaps other services, you may need to provide configuration information for country name aliases (see Section 15.1.3.1.3) and address formats (see Section 15.1.3.1.4).

Note: All location services configuration information, except YP category information, is maintained internally (by Oracle9*i*AS Wireless) in an XML configuration file named `site_cfg_bootstrap.xml`. However, you are encouraged *not* to modify that file directly; instead, use the webtool interface to modify configuration information.

15.1.3.1.2 Provider Configuration An ordered list of providers is configured with the following parameters:

- Provider Name: the provider name, which serves as an ID
- Provider Impl Class: the class implementing the proxy for this provider (for translation and communication with the provider)
- URL: the static URL prefix used to access the provider
- User Name: a user name as determined by the provider
- Password: the password to be used in combination with the user name
- Parameters: any parameters required to customize and configure the provider proxy
- ISO Locales: a semicolon-delimited list of country IDs (as specified in the country name alias list, described in Section 15.1.3.1.3)
- Corporate URL: the corporate URL of the provider (used as an advertisement)
- Service Version: the service version for the provider that this proxy uses
- Corporate Logo URL: the corporate logo URL of the provider (used as an advertisement)

15.1.3.1.3 Country Name Alias Configuration The country name alias configuration relates country names and synonyms to a single standard identifier for a given country. This standard identifier should be the ISO name (US for US, DE for Germany, and so on), although you can specify other identifiers.

The aliases are used in combination with the `oracle.panama.spatial.core.ruleengine.RuleEngineImpl` provider selection framework implementation. Each provider is configured for a set of countries, specified by their IDs. For example, when a service request is made, for example to geocode an address in the *United States*, the country alias table is consulted to find the standard ID *US*. Subsequently, only providers with *US* in their list of covered countries are tried.

If a country name is used which is not configured as a known alias for some country ID, the ID *unknown* is used, instead. In this case, providers with *unknown* in the covered country list are tried.

If the simple provider selection framework implementation (`oracle.panama.spatial.core.ruleengine.SimpleRuleEngineImpl`) is used, country aliases are not required for provider selection.

15.1.3.1.4 Address Format (International) Configuration The address format configuration is used to specify international address formats. The `oracle.panama.spatial.intladdress` package in the API uses this list to determine which components are part of an address (US, French, German, Chinese, and so on) and how they are presented for input and output.

The international address framework is configured with a list of address formats in the repository, accessible through the webtool. This configuration specifies all components of an address, aliases for the components, and mappings to standard concepts such as *city*, *state*, and *street name*. The format of the textual representation is also configured, to determine such things as:

- How is the address usually divided into separate lines?
- In which sequence do the components occur?
- Which components are optional, and which are required?

This approach requires that users specify a country-specific format for addresses, in order to view and enter addresses in that format. Otherwise, for example, the system cannot know whether to ask a user to specify a state or province before the country.

The benefits of this approach include the following:

- Users see a form that exactly matches the desired address format for mailed letters.
- The system can better analyze addresses when each component is known separately and meaningfully identified, rather than simply being included somewhere in *first line*, *second line*, and *last line*.
- An application looks more professional if it automatically adapts to the local address format, both for input and output of addresses.
- Outside the US, customers are much more impressed when presented with their local address format, as opposed to the US format.
- The application does not have to be rewritten for different countries. Everything is handled automatically by the framework

Several international address formats are supplied. Two examples are as follows.

For the US:

```
{name}
{house number/house} {street}[ Apt {apt}]
{city} {state} {postal code}[-{postal code ext}]
{country}
```

For Germany:

```
{Name/name}
{Strasse/street/first line} {Hausnummer/house}[ Wohnung {Wohnung/apt}]
{PLZ/postal code} {Stadt/city}
 [{Bundesland/state}]
{Land/country}
```

Syntax notes:

- { } (braces) enclose an address component.
- / (slash) separates alternative aliases within a component.
- [ ] (brackets) enclose optional elements.
- Anything outside braces other than brackets is taken as quoted from an address (such as *Apt* in *123 Main Street Apt 4*).

For programming information and examples relating to international address formats, see Section 15.2.2.1.1.

15.1.3.2 Provider Selection Logging

The provider selection framework implementation logs selection, success, and failure of providers on the *i*AS servlet container console or in the Oracle9*i*AS Wireless log file (for example, `sys_panama.log`). For example, you can look for events such as the following:

- The multiplexers for geocoding, mapping, and so on (other types of services) have been initialized.
- The provider selection framework implementation has been initialized.
- The proxies for geocoding, mapping, and so on (other types of services) have been initialized.
- A specific provider has been tried.
- A specific provider has failed.
- A specific provider has succeeded.
- All providers have failed.

15.1.4 Geocoding Services

The geocoding API provides the geographic location of a given address. For a user of Oracle9*i*AS Wireless, an address is the most common way to specify a location. However, for finding restaurants in close vicinity or providing driving directions, the text representation of an address may not be useful unless it is first geocoded, that is, translated to geographic coordinates.

The address to be geocoded has a textual representation like that from a standard mailed letter. The result returned is the longitude / latitude corresponding to the address. For example, the input to geocoding might include the following:

- firmName: "oracle"
- firstLine: "1 Oracle Drive"
- secondLine: ""
- lastLine: "Nashua NH 03062"
- matchMode: "tight"

In this example, the result is: Point(x = -71.455, y = 42.7117)

Because a user might specify an ambiguous address, the `GeocodeResult` contains an array of Location objects instead of a single object.

15.1.4.1 Geocoding API

This section describes the geocoding API for location application components.

Two of the following classes, `Point` and `Location`, are used by the whole API and are not specific to geocoding. However, they are described here because they represent components central to the geocoding service, both for input and output.

15.1.4.1.1 Point Class The `Point` class defines a longitude/latitude coordinate point. Additional values for a label and a radius can be used for representing a point on a map. The label and radius are not used by any other functions than map display.

15.1.4.1.2 Location Class The `Location` class defines a location with address and longitude/latitude. If the location object is constructed using `firstLine`, `secondLine`, and `lastLine`, then some external providers might not correctly identify the city or state, because `lastLine` can contain city, state, and postal code in a country-specific and relatively flexible format.

If no specific substring can be identified as the component representing the city, the city is "unknown". In this case, the API itself might not try complex analysis, but instead leave this task to the experts, that is, the external geocode providers.

15.1.4.2 Geocoder Interface

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

15.1.5 Location Marks

Due to the limitations of certain mobile devices such as telephones, it is difficult to input/display lengthy alphanumeric strings. A **location mark** stores a piece of spatial information identified by a concise, easy-to-understand name. For example, "My home" might be the name of a location mark, while the underlying spatial information might be "123 Main Street, Somewhere City, CA, 12345; Lon = -122.42, Lat = 37.58".

[preceding for when rectangles are allowed] For example, "My home" and "Downtown San Francisco" might be the names of two location marks, while the underlying spatial information might be "123 Main Street, Somewhere City, CA, 12345; Lon = -122.42, Lat = 37.58" for the first location mark, and "The rectangle geometry within bounding points (Lon/Lat = -122.49, 37.79) and (Lon/Lat = -122.41, 37.74)" for the second location mark.

Location marks allow users to avoid inconvenient string input on mobile devices. Users can manage their location marks on a desktop and then access them by referring to their names from mobile devices. Today's location-aware applications typically just use a point location (such as an address or a road intersection). In this case, the spatial information can be provided by geocoding. In this release, a location mark must be a point. However, in a future release it is planned to allow a location mark to be something other than a point; for example, it could be the current position associated with automatic mobile positioning or a region defined by region modeling.

Location marks also allow users to try "what-if" scenarios: to make an application behave as if they were in a location different from their default or current location. For example, a user of an entertainment services application might actually be in Boston now, but will be traveling to San Francisco in a few days. This person could set a location mark in San Francisco as the default, and be presented with information relevant to the San Francisco area.

Each user has personalized location marks, which are stored in the Oracle9*i*AS Wireless repository.

Location marks are created using the `LocationMark` class. Users can also create location marks by logging into the *i*AS Personalization Portal, clicking the **LocationMarks** tab, and clicking **Create**.

For information about using a location mark to enable mobile positioning, see Section 15.3.1.

15.1.6 LOCATIONMARK Table

A new table named LOCATIONMARK is added to the Oracle9*i*AS Wireless repository schema. This table contains detailed information about each location mark, including the user associated with each location mark. For example, several users might have a location mark named *Office* but with a different location for each.

Table 15–1 lists the columns in the LOCATIONMARK table.

Table 15–1 LOCATIONMARK Table columns

Column Name	Type
objectId_	NUMBER(10)
name	VARCHAR2(32)
userId	NUMBER(10)

Table 15–1 LOCATIONMARK Table columns (Cont.)

Column Name	Type
longitude	NUMBER
latitude	NUMBER
addrline1	VARCHAR2(256)
addrline2	VARCHAR2(256)
addrlastline	VARCHAR2(256)
block	VARCHAR2(256)
city	VARCHAR2(256)
country	VARCHAR2(256)
county	VARCHAR2(256)
firmname	VARCHAR2(256)
pcode	VARCHAR2(32)
pcode_ext	VARCHAR2(16)
state	VARCHAR2(256)
matchmode	VARCHAR2(32)
description	VARCHAR2(256)

15.1.7 Mapping Services

The mapping API provides functions for creating map images for any of the following:

- A single point (such as an address or a location mark)
- Multiple points (such as several addresses or location marks)
- A complete route
- A single driving maneuver

The mapping API lets you specify the size (resolution) of the map and the image format. Image transformation is also provided through the `oracle.panama.spatial.imagex` class (discussed in Section 15.1.8.3). For example, if the provider only supports GIF format but the user requires WBMP format, the `imagex` API can perform the transformation.

Mapping capabilities can be made visible to users as a purely mapping application or as part of a routing application. In a routing application, the mapping of routes and driving maneuvers is performed by the routing provider. For information about routing services, see Section 15.1.8.

15.1.8 Routing Services

The routing API provides routing (driving directions) information based on a start point, an end point, and optionally a list of intermediate via points. All points are specified as longitude/latitude pairs or addresses.

The routing result consists of a set of maneuvers. A **maneuver** corresponds to a driving instruction, such as "turn left onto I-93" or "bear right and merge to Route 3". The routing result also includes estimated driving time and distance. Optionally, maps and route coordinates can be requested.

15.1.8.1 Routing Settings

Routing can be influenced by preferences or requirements, called routing options. These options are combined in a set, called **routing settings**. There are two types of routing options: basic options and secondary options.

Basic options include:

- Whether maps (images) are requested
- Whether geometries (route coordinates) are requested

Secondary options include:

- Optimization method, such as shortest distance or shortest driving time
- Route properties to avoid, such as toll roads, ferry lanes, or limited-access highways
- Map sizes

Secondary options can be mandatory or preferred:

- If a secondary option is mandatory but not supported by the provider, the API will automatically fail over to the next provider.
- If a secondary option is preferred but not supported by the provider, the API will not check to see if other providers support the option.

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

- 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

15.1.8.2 Routing Results

The application can query the following components of a returned route:

- List of maneuvers
- Total distance
- Total estimated driving time
- Overview map

An overview map shows the source and destination, with the route highlighted. Figure 15–5 shows an overview map for the driving direction from New York City to Washington, DC.

Figure 15–5 Overview Map

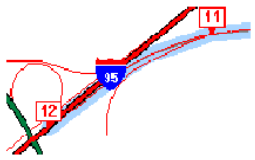
A set of maneuvers (driving directions) is returned as part of the routing result. Each maneuver corresponds to a driving instruction and contains the following information:

- Textual narrative

- Distance traveled during or prior to this maneuver ("After how many miles do I have to make this right turn?")
- Detailed maneuver map
- Geometry (list of coordinate points, longitude/latitude)

Figure 15–6 shows a maneuver map for merging onto Interstate Route 95. The narrative might be "Continue on ramp at sign reading 'Exit 5 I-95 South Del. Tpke. To Baltimore' and go southwest for 0.4 miles."

Figure 15–6 Maneuver Map



Maps of the complete route or maneuvers can be requested as Java Image objects or as Strings representing a URL.

15.1.8.3 Map Options and Transformation Requirements

The application can affect the size of the map. Because different devices (for example, a desktop web browser, a Palm device, and a phone) may need specific image formats (for example, GIF, BMP, or WBMP) and because most providers support only a single format, the application currently is responsible for image transformation between formats.

Transformation support is provided in the `oracle.panama.spatial.imagex` class. This class provides functions for transforming images to different image formats, sizes, and orientations. You can use this class to perform the following operations on images:

- Convert between BMP, WBMP and GIF
- Scale image size
- Rotate (90°, 180°, 270°)
- Flip (vertically, horizontally, diagonally and antidiagonally)

15.1.8.4 Support for Multiple Languages

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

15.1.8.5 Routing API

This section describes the routing API for location application components.

15.1.8.5.1 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`.

15.1.8.5.2 RoutingSettings Class The `RoutingSettings` class defines a set of options passed to routing. There are two types of routing options: basic and secondary.

Basic options include whether or not to request a map or a geometry. Basic options can be specified in the constructor of a `RoutingSettings` object.

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.

Whether or not the secondary option is mandatory is defined 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 assumed.

15.1.8.5.3 RoutingResult Class The `RoutingResult` class defines the routing results, which are described in Section 15.1.8.2.

15.1.8.5.4 Maneuver Class The `Maneuver` class defines a single maneuver in a route (see Section 15.1.8.2 for the maneuver attributes).

15.1.9 Business Directory (Yellow Page) Services

Business directory (Yellow Page, or YP) services provide lists of businesses in a given area and matching a specified name or category.

Existing providers use YP services with different interfaces. Specifically, they all have different YP categories, and even different hierarchical structures. The

categories might be organized in a flat list or in a hierarchy of categories and subcategories. A hierarchy tree might be deep or shallow, with a high or low fanout, and might be balanced or unbalanced.

To unify the service of different providers, the Oracle business directory services use a custom hierarchy that the Oracle9*i*AS Wireless developer defines in an XML file. Each leaf in this hierarchy has a reference to a category of one or more providers. Non-leaf nodes might also have such references. This custom hierarchy defines preferred categories first. Subsequently, the carrier using Oracle9*i*AS Wireless tries to match these categories to semantically similar categories supported by external providers.

The customized hierarchy with the references to external providers' categories is represented in an XML file that stores hierarchical and ordered structures. Representing order in the category hierarchy can account for the popularity of different categories. For example, on a device with a limited screen size, an application might restrict the choices among the most popular categories.

15.1.9.1 Different Approaches Among Yellow Pages Providers

Several providers offer YP services on the web; however, the approaches taken by these providers differ significantly and do not offer a uniform interface. Furthermore, the respective approaches are not final in their methodology and can be expected to change.

A unifying pattern in the various approaches is that businesses are categorized by subject and location. The location component is well understood in that either a ZIP code or the combination of a city and state can be used to determine the location.

The categorization of businesses, on the other hand, is not uniformly implemented. Some providers offer a flat list of categories, user-selected by simple substring matching. Another approach is a 3-level or 4-level hierarchical organization of subcategories, often with a fanout of 20 to 50, sometimes more than 100. A user might start the hierarchy traversal at the root of the hierarchy (by default). Alternatively, a user might enter a keyword that is matched to an appropriate starting point within the hierarchy. Such keyword matching might go beyond simple substring search and result in more intelligent choices.

15.1.9.2 Business Directory Category Configuration

Support for business categories and the hierarchy of categories is provided through an XML configuration file. (You should view and modify business directory provider information using the webtool; however, you must view and modify business directory category information using the XML file.)

The category hierarchy definition file in Example 15–1 represents the custom hierarchy of business directory categories. Each category can have any number of subcategories. There is no restriction to the level of nesting. A category can be linked to multiple business directory content providers. The flexibility allowed by this file accommodates the different approaches of various business directory service providers, as discussed in Section 15.1.9.1.

Example 15–1 Business Directory Category Hierarchy Definition File

```
<?xml version="1.0" standalone="yes"?>
<Categories>
  …
  <Category
    CategoryName  = "Berry crops">
    <Provider
      Name  = "…"
      Parameter  = "…"/>
      <Category
      CategoryName  = "Cranberry farm">
      <Provider
        Name  = "…"
        Parameter  = "…"/>
    </Category>
  </Category>
  …
  <Category
    CategoryName  = "Ornamental nursery products">
    <Provider
      Name  = "…"
      Parameter  = "…"/>
    <Category
      CategoryName  = "Florists' greens and flowers">
      <Provider
        Name  = "…"
        Parameter  = "…"/>
    </Category>
      <Category
      CategoryName  = "Bulbs and seeds">
      <Provider
        Name  = "…"
        Parameter  = "…"/>
    </Category>
  </Category>
  <Category
    CategoryName  = "Crops grown under cover">
```

```
    <Provider
      Name  = "…"
      Parameter  = "…"/>
    <Category
      CategoryName  = "Mushrooms grown under cover">
      <Provider
        Name  = "…"
        Parameter  = "…"/>
    </Category>
  </Category>
  …
</Categories>
```

15.1.9.3 Business Directories (Yellow Pages) API

The application developers can traverse the category hierarchy by using the functions in the `YPFinder` interface. For any resulting category, the following can be requested:

- List of businesses
- List of direct subcategories
- List of direct or indirect subcategories containing a substring

15.1.9.3.1 YPFinder Interface The `YPFinder` interface defines how an application programmer accesses the YP service. An object of a class implementing this interface is returned by the `SpatialManager`.

An object of this class lets the user query:

- Businesses in a state
- Businesses in a city
- Businesses in a postal code
- Businesses in a radius around a center
- The closest *n* businesses around a center

In each of these region types, businesses can be found:

- Matching a given business name or keyword
- Matching a given category
- Matching both a given business name or keyword and a given category

- Matching a keyword in either a business name or category

15.1.9.3.2 YPCategory Class The `YPCategory` class defines a single category that is part of the hierarchy. This class lets users access businesses in the category. It also lets users find subcategories of the category; specifically, you can find:

- All the direct subcategories
- All direct or indirect subcategories matching a keyword
- A subcategory with a given name

One of the most popular applications probably is to find subcategories of the root matching a given keyword.

15.1.9.3.3 YPBusiness Class The `YPBusiness` class defines a single business. It represents an address (`Location` interface) that also has a telephone number, a description, and a list of categories it matches. You can get all businesses in a category or all categories for each of these businesses. For example, a given bookstore might be both in the categories *book store* and *cafe*.

15.1.10 Traffic Services

The traffic API provides information about conditions that can affect traffic flow on road networks in major metropolitan areas. These areas are typically further divided into smaller areas, such as downtown, metro West, metro East, and so on. Real-time traffic reports update conditions in short time periods (such as every 5 minutes), thus providing information that is important for fleet management as well as personal navigation.

The major components of traffic reports are incidents. An **incident** is an event that will probably affect the flow of traffic. Examples of incidents are accidents, construction activity, and traffic congestion (normal or unexpected). Each incident includes such information as the type of incident, where the incident occurred (such as the route number, the location, or the region), the direction along the route (such as northbound), the expected delay, and the length of the traffic backup.

For the current release, the following kinds of queries are supported for incident-based traffic information:

- City-level query: return traffic incidents in the entire city.
- Route-level query: return traffic incidents on the specified route in a city.
- Longitude/latitude (point) or address plus radius-level query: return traffic incidents in the requested circular area.

Examples of traffic queries include returning the traffic report for:

- A metropolitan area (such as Boston)
- A route in a metropolitan area (such as I-93 South in Boston)
- A planned route (such as from Nashua, NH to Boston, MA), returned as a collection of (route, city)
- A mobile range consisting of a location (longitude, latitude) and a radius from the location
- The vicinity of a given address (such as One Oracle Drive, Nashua, NH 03062)

The traffic API processes requests and returns responses. The requests and responses can be in Java or XML format. Section 15.1.10.2 provides examples of an XML request and response in XML format. Section 15.1.10.3 describes the traffic Java API.

Note: For the current release, no traffic service providers are included in the sample configuration files.

15.1.10.1 Traffic Report Caching

Traffic report information is cached at the city level. The first time that a traffic report on a city is fetched, the report is written to the traffic report cache. The cached report is considered invalid after a maximum cache age time (for example, 15 minutes), which can be set using the *i*AS webtool.

A network round-trip operation to the traffic service provider is required to update the cached traffic report for the city. The cached report is updated only when both of the following conditions are true:

- A query is made for the city or for any Spatial geometry (route or point with radius) that is in or partially in the city (that is, where the queried geometry spatially interacts with the city's geometry).
- The cached report for the city is older than the maximum cache age time.

15.1.10.2 Traffic XML Requests and Responses

Example 15–2 shows a city-level request in XML format for traffic information for Boston.

Example 15–2 Traffic Request for Boston

```
<?xml version="1.0" encoding="UTF-8"?>
<traffic_request>
  <query_list>
    <query_info
query_type="city_level_query"
city_name="boston"
state_name="MA"
country_name="US"
    />
  </query_list>
</traffic_request>
```

Example 15–3 shows a response in XML format for traffic information for Boston.

Example 15–3 Traffic Response for Boston

```
<?xml version="1.0"  encoding="UTF-8"?>
<traffic_response>
 <report_list>
  <traffic_report>
    <provider
    name="Trafficstation"
    covered_city_name="Boston"
    state_name="MA"
    country_name="US"/>
    <report_time month="6" day="19" year="2001" hour="5" minute="28" meridian =
      "PM"/>
      <unit distance_unit="MILES" time_unit="MINUTE"/>
    <incident_list>
      <incident id = "1">
        <incident_type>ACCIDENT</incident_type>
        <description>CAR ACCIDENT</description>
        <route type = "Interstate" name = "I-93" direction = "SOUTH"/>
        <geo_location  longitude = "-71.0607" latitude  = "42.3659" radius =
           "5.0"/>
        <location_range>
          <at_location>EXIT 26</from_location>
        </location_range>
        <time_range>
          <from_time  month = "6" day = "19" year = "2001" hour = "5" minute =
             "28" meridian = "PM"/>
          <to_time  month = "6" day = "19" year = "2001" hour = "5" minute =
             "28" meridian = "PM"/>
         </time_range>
```

```
        <severity>HEAVY</severity>
        <speed>15.0</speed>
        <impact>EXPECT DELAY</impact>
        <advice>TAKE LEFT LANE</advice>
      </incident>
      <incident id = "2">
        <incident_type>CONSTRUCTION</incident_type>
        <description>REGULAR MAINTENANCE</description>
        <route type = "Interstate" name = "I-95" direction = "NORTH"/>
        <geo_location  longitude = "-71.3555" latitude  = "42.3601" radius =
          "30.0"/>
        <location_range>
          <at_location>EXIT 36</at_location>
        </location_range>
        <time_range>
          <at_time  month = "6" day = "19" year = "2001" hour = "5" minute =
             "28" meridian = "PM"/>
         </time_range>
        <severity>MINOR</severity>
        <speed>35.0</speed>
        <impact>EXPECT DELAY</impact>
        <advice>USE I-495</advice>
      </incident>
    </incident_list>
  </traffic_report>
 </report_list>
</traffic_response>
```

15.1.10.3 Traffic Java API

This section describes the traffic Java API for location application components.

15.1.10.3.1 CityInfo Class The `CityInfo` class provides the city name, state name, and country name for a city. A common use of this class is to create a `CityInfo` instance with city name, state name (optional), and country name, and pass it to the query for a traffic report at city level, route level, or point and radius level with a city.

15.1.10.3.2 City Interface The `City` interface provides information about a city from a specified service provider. The information includes the city name, state name, country name, and information about routes. A `City` instance could be obtained from the `TrafficReport` interface.

15.1.10.3.3 RouteInfo Class The `RouteInfo` class provides the name and type for a route. A common use of this class is to create a `RouteInfo` instance and pass it to the query for a traffic report at route level.

15.1.10.3.4 TrafficRoute Interface The `TrafficRoute` interface provides information for a route from a specified service provider. The information includes the route name, route type, geometry that represents the route, and the city name. A `TrafficRoute` instance could be obtained from the `TrafficIncident` interface.

15.1.10.3.5 TrafficReport Interface The `TrafficReport` interface provides information for an incident-based traffic report, such as the report time, the number of incidents, the provider's information, the city, and the incidents. A report could then be created to show to users or administrators of the application.

15.1.10.3.6 TrafficIncident Interface The `TrafficIncident` interface provides information for a traffic incident, such as the severity, type, description, route and direction on which the occurred, location, time range, impact, and advice.

15.1.10.3.7 TrafficReporter Interface The `TrafficReporter` interface provides functions that return a traffic report based on different queries. The following kinds of queries are supported:

- Given the information about a city (city name, state name [optional], country name), return the report.
- Given the information about a route (with or without direction) and the city where the route is located in, return the report.
- Given the longitude / latitude coordinates of a point and the radius, return the report for the area.
- Given the address of a location and the radius, return the report for the area.

When using `SpatialManager.createLocation()` to get an instance of `Location`, you must specify the city name and country name. Do not use the `LastLine` attribute to combine these pieces of information. Set the value of the `Point` geometry to null to avoid automatic geocoding.

15.1.10.3.8 TrafficCityManager Interface The `TrafficCityManager` interface provides two functions, one to obtain all the cities for which traffic information is provided, and the other to obtain the routes info for a given city. A common use of these functions is to call them to create a drop-down list of cities and routes supported by the application.

15.1.10.4 Traffic Service Configuration

After the region modeling data and city coverage data has been loaded into the repository during the Oracle9*i*AS Wireless installation, you can add traffic providers and supported cities for a provider.

15.1.10.4.1 Adding a Traffic Provider To add support for a new traffic service provider, follow these steps:

1. Using the *i*AS webtool, set the traffic provider information and the traffic report cache time.
2. For each supported city of this new provider, use the region modeling tool (described in Section 15.4) to check if there is an entry in the CITY table for that city, including a valid GEOMETRY column value. If there is not an entry for the city, including its geometry, add an entry.
3. Get and note the ID of this city.
4. Use SQL*Plus connect to the Oracle9*i*AS Wireless repository.
5. For each city to be supported for this traffic service provider, set the value of the COVERED_BY_TRAFFIC column in the CITY_COVERAGE table to 'Y', and use the value of city's ID to perform the update. For example:

   ```
   UPDATE city_coverage SET covered_by_traffic = 'Y' WHERE id = 12345;
   COMMIT;
   ```

   If there is not already an entry in the CITY_COVERAGE table for this city, add a row and set the value of the COVERED_BY_TRAFFIC column to 'Y', and be sure that the ID value in this table is the same as the ID value for the city in the CITY table. For example:

   ```
   INSERT INTO city_coverage (id, name, state_name, country_name,
        covered_by_traffic) VALUES (10750, 'BOSTON', 'MA', 'US', 'Y');
   COMMIT;
   ```

15.1.10.4.2 Adding a Supported City for a Provider To add support for a new city for an existing traffic service provider, follow these steps:

1. For each supported city of this new provider, use the region model tool (described in Section 15.4) to check if there is an entry in the CITY table for that city, including a valid GEOMETRY column value. If there is not an entry for the city, including its geometry, add an entry.
2. Get and note the ID of this city.

3. Use SQL*Plus connect to the Oracle9*i*AS Wireless repository.
4. If there is an entry in the CITY_COVERAGE table for this city, set the value of the COVERED_BY_TRAFFIC column in the CITY_COVERAGE table to 'Y', and use the value of city's ID to perform the update. For example:

```
UPDATE city_coverage SET covered_by_traffic = 'Y' WHERE id = 12345;
COMMIT;
```

   If there is not already an entry in the CITY_COVERAGE table for this city, add a row and set the value of the COVERED_BY_TRAFFIC column to 'Y', and be sure that the ID value in this table is the same as the ID value for the city in the CITY table. For example:

```
INSERT INTO city_coverage (id, name, state_name, country_name,
     covered_by_traffic) VALUES (10750, 'BOSTON', 'MA', 'US', 'Y');
COMMIT;
```

15.2 Developing Location-Based Applications

You can develop a location-based application by using either of the following approaches:

- Creating Java Server Page (JSP) files that contain MobileXML and/or HTML tags and that include custom Oracle-supplied tags
- Writing a Java program (specifically, an adapter)

Creating JSP files is often easier and more convenient that writing an adapter; however, writing an adapter gives you greater flexibility and control over the program logic. This section describes both approaches.

15.2.1 Creating Java Server Pages

If you do not need to write an adapter, you can create Java Server Pages (JSP files) to provide location-based capabilities to users.

This section provides detailed information about the Oracle-supplied tags that you can use. Each reference section includes an example.

Table 15–2 groups the JSP tags for location services by the type of application for which the tag is useful, and briefly describes the information specified by the tag.

Table 15–2 JSP Tags for Location Services

Type of Application	Tag Name	Specifies
Geocoding, Mapping, Routing	address	An address to be geocoded, located on a map, or used as the start or end address of a route or as the center for a business directory query
Mapping	map	A map with a specified resolution, including one or more points or routes, or showing a driving maneuver or a complete route
Routing	route	A route with a specified map resolution
Routing	iterateManeuvers	A collection of driving maneuvers, presented individually
Business directory	businesses	A collection of businesses that share one or more attributes
Business directory	iterateBusinesses	A collection returned by the `businesses` tag, presented individually
Business directory	category	A business category (for example, `Dealers`)
Business directory	iterateCategoriesMatchingKeyword	A collection of categories that match a specified keyword value, presented individually
Business directory	iterateChildCategories	A collection of immediate child subcategories, presented individually

These tags must be used with a prefix, which must be specified in the JSP file. The following example defines the `loc` prefix, which is used in other examples of specific tags:

```
<%@ taglib uri="LocationTags" prefix="loc" %>
```

The following example shows the `loc` prefix used with the `address` tag:

```
<loc:address name="hq" type="oracle.panama.model.Location"
    businessName="Oracle Headquarters" firstLine="500 Oracle Parkway"
    city="Redwood City" state="CA" postalCode="94065" country="US"/>
```

The following sections (in alphabetical order by tag name) provide reference information for all the parameters available for each tag: the parameter name, a

description, and whether or not the parameter is required. If a parameter is required, it must be included with the tag. If a parameter is not required and you omit it, the interpretation is performed by the service provider.

Short examples are provided in the reference sections for JSP tags, and more comprehensive examples are provided in Section 15.2.1.10.

and evaluated at run time:

- If a parameter is required, it must be included with the tag. If a parameter is not required and you omit it, the interpretation is performed by the service provider.

15.2.1.1 address

The `address` tag specifies an address to be geocoded, located on a map, or used as the start or end address of a route or as the center for a business directory query.

Table 15–3 lists the address tag parameters. (See Section 15.2.1 for an explanation of the information provided.)

Table 15–3 Address Tag Parameters

Parameter Name	Description	Required
name	Name for the returned address object. Example: `hardware_1`.	Yes
type	Type of object. Must be: `oracle.panama.model.Location`	Yes
businessName	Descriptive name of the business or other entity at the address. Example: `Mike's Hardware`	No
firstLine	Street address.	No
city	City name.	No
state	2-character state (US) or province (Canada) code.	No
postalCode	Postal code.	No
country	Country name.	No

The following example of the `address` tag specifies an address (for a store named Mike's Hardware) to be geocoded.

```
<loc:address
  name = "hardware_1"
  type = "oracle.panama.model.Location"
  businessName = "Mike's Hardware"
```

```
firstLine = "22 Monument Sq"
city = "Concord"
state = "MA"
postalCode = "01742"
country = "US"  />
```

15.2.1.2 businesses

The `businesses` tag specifies a collection of businesses that share one or more attributes.

Table 15–4 lists the businesses tag parameters. (See Section 15.2.1 for an explanation of the information provided.)

Table 15–4 Businesses Tag Parameters

Parameter Name	Description	Required
name	Name for the returned object. Example: `mikes_hardware_stores`	Yes
type	Type of object. Must be: `java.util.Collection`	Yes
businessName	Descriptive name of the business or other entity at the address. Example: `Mike's Hardware`	No
categoryID	Business services category variable name. Example: `Automotive`.	No
keyword	Any string to search for in the `name` or `categoryID`. Example: `French`	No
city	City name.	No
state	2-character state (US) or province (Canada) code.	No
postalCode	Postal code.	No
country	Country name	No
centerID	A point variable name (such as for an address) to be used as the center point from which to start searching. If you specify `centerID`, you must also specify `radius` or `nearestN`.	No
radius	Length (in meters) of the radius of the circle in which to search. If you specify `radius`, you must also specify `centerID`.	No
nearestN	Maximum number of nearest results that satisfy the query requirements (for example, to find the 3 nearest banks to a hotel or the user's current position). If you specify `nearestN`, you must also specify `centerID`.	No

The following example of the businesses tag specifies all businesses named Borders in the state of California in the United States. The use of the map tag to enclose the businesses tag causes a map to be created that includes and labels each Borders bookstore.

```
<loc:map name="map1" type="oracle.panama.spatial.jsptags.beans.Map"
       xres="1000" yres="500">
   <loc:businesses name="bord" type="java.util.Collection"
businessName="Borders"
        country="US" state="CA"/>
</loc:map>
```

15.2.1.3 category

The category tag specifies a business category (for example, Dealers).

Table 15–5 lists the category tag parameters. (See Section 15.2.1 for an explanation of the information provided.)

Table 15–5 Category Tag Parameters

Parameter Name	Description	Required
name	Name for the returned object. Example: cat_dealers	Yes
type	Type of object. Must be: oracle.panama.spatial.yp.YPCategory	Yes
parentCategory	Name of the object containing the specification of the parent category (created previously using the category tag). If not specified, the root is assumed.	No
categoryName	Name of the category. Example: Dealers.	Yes

The following example uses two category tags. The first category tag creates an object named cat_auto that specifies a category named Automotive. The second category tag creates an object named cat_dealers that specifies a category named Dealers that is a child of the cat_auto (Automotive) parent category.

```
<mt:category name="cat_auto" type="oracle.panama.spatial.yp.YPCategory"
   categoryName="Automotive" />
<mt:category name="cat_dealers" type="oracle.panama.spatial.yp.YPCategory"
   parentCategory="cat_auto" categoryName="Dealers" />
```

15.2.1.4 iterateBusinesses

The `iterateBusinesses` tag presents individually the businesses in a collection returned by the `businesses` tag.

Table 15–6 lists the `iterateBusinesses` tag parameters. (See Section 15.2.1 for an explanation of the information provided.)

Table 15–6 IterateBusinesses Tag Parameters

Parameter Name	Description	Required
name	Name for the returned object. Example: `iter_borders`	Yes
type	Type of object. Must be: `oracle.panama.model.Point`	Yes
collection	Name for the returned collection. Example: `bord`	Yes

The following example creates a collection of all Borders stores in California and then uses the `iterateBusinesses` tag to present each Borders store location in the collection.

```
<loc:businesses name="bord" type="java.util.Collection" businessName="Borders"
        country="US" state="CA"/>
<loc:iterateBusinesses name="iter_borders" type="oracle.panama.model.Point"
    collection="bord" />
```

15.2.1.5 iterateCategoriesMatchingKeyword

The `iterateCategoriesMatchingKeyword` tag creates a collection of categories that match a specified keyword value, and presents the categories individually.

Table 15–7 lists the `iterateCategoriesMatchingKeyword` tag parameters. (See Section 15.2.1 for an explanation of the information provided.)

Table 15–7 IterateCategoriesMatchingKeyword Tag Parameters

Parameter Name	Description	Required
name	Name for the returned object. Example: `hardware_stores`	Yes
type	Type of object. Must be: `oracle.panama.spatial.yp.YPCategory`	Yes
parentCategory	Name of the object containing the specification of the parent category (created previously using the `category` tag).	No

Table 15–7 IterateCategoriesMatchingKeyword Tag Parameters (Cont.)

Parameter Name	Description	Required
keyword	Word or phrase to be searched for in the parent category name, or in all category names if `parentCategory` is not specified.	Yes

The following example shows the `category` tag used to create an object named `cat_auto` that specifies a category named `Automotive`. The `iterateCategoriesMatchingKeyword` tag is then used to create an object named `key` that contains each category containing `Dealers` under the Automotive parent category, and the fully qualified name of each returned category is displayed.

```
<mt:category name="cat_auto" type="oracle.panama.spatial.yp.YPCategory"
   categoryName="Automotive" />
<mt:iterateCategoriesMatchingKeyword name="key"
   type="oracle.panama.spatial.yp.YPCategory"
   keyword="Dealers" parentCategory="cat_auto">
      >>> <%= key.getFullyQualifiedName() %>
</mt:iterateCategoriesMatchingKeyword>
```

15.2.1.6 iterateChildCategories

The `iterateChildCategories` tag specifies a collection of immediate child subcategories, presented individually.

Table 15–8 lists the `iterateChildCategories` tag parameters. (See Section 15.2.1 for an explanation of the information provided.)

Table 15–8 IterateChildCategories Tag Parameters

Parameter Name	Description	Required
name	Name for the returned object. Example: `hardware_stores`	Yes
type	Type of object. Must be: `oracle.panama.spatial.yp.YPCategory`	Yes
parentCategory	Name of the object containing the specification of the parent category (created previously using the `category` tag).	No

The following example of the `iterateChildCategories` tag presents each immediate child category under the `automotiveDealers` category.

```
<loc:category name="automotiveDealers"
  type="oracle.panama.spatial.yp.YPCategory" categoryName="Dealers">
<loc:category name="automotive" type="oracle.panama.spatial.yp.YPCategory"
  categoryName="Automotive"/>
</loc:category>

<loc:iterateChildCategories name="cat"
  type="oracle.panama.spatial.yp.YPCategory" parentCategory="automotiveDealers">
<%= cat %>
```

15.2.1.7 iterateManeuvers

The `iterateManeuvers` tag creates a collection of driving maneuvers, and it presents the maneuvers individually.

Table 15–9 lists the `iterateManeuvers` tag parameters. (See Section 15.2.1 for an explanation of the information provided.)

Table 15–9 IterateManeuvers Tag Parameters

Parameter Name	Description	Required
name	Name for the returned object. Example: `eManeuver`	Yes
type	Type of object. Must be: `oracle.panama.spatial.jsptags.beans.route`	Yes
routeID	Name of the route for which to present the driving maneuvers.	Yes

The following example creates a route named `myRoute` between two Oracle offices, displays a map of the route, and presents each driving maneuver (using the `iterateManeuvers` tag and the `getMap` and `getNarrative` function calls).

```
<loc:route name="myRoute" type="oracle.panama.spatial.jsptags.beans.Route"
       xres="800" yres="600">
   <loc:address
        name="NEDC"
        type="oracle.panama.model.Location"
        businessName="NEDC"
        firstLine="1 Oracle Dr"
        city="Nashua"
        state="NH"
        postalCode="03062"
        country="US"/>
   <loc:address
```

```
        name="HQ"
        type="oracle.panama.model.Location"
        businessName="HQ"
        firstLine="500 Oracle Parkway"
        city="Redwood City"
        state="CA"
        postalCode="94065"
        country="US"/>
</loc:route>

<img src="<%= myRoute.getMap() %>">

<loc:iterateManeuvers name="aManeuver"
   type="oracle.panama.spatial.jsptags.beans.Maneuver" routeID="myRoute">
      <a href="<%= aManeuver.getMap() %>">
        <%= aManeuver.getNarrative() %>
      </a>
</loc:iterateManeuvers>
```

15.2.1.8 map

The `map` tag specifies a map with a specified resolution and showing one of the following:

- One or more points
- A route
- A driving maneuver

Table 15–10 lists the `map` tag parameters. (See Section 15.2.1 for an explanation of the information provided.)

Table 15–10 Map Tag Parameters

Parameter Name	Description	Required
name	Name for the returned object. Example: `myMap`	Yes
type	Type of object. Must be: `oracle.panama.spatial.jsptags.beans.Map`	Yes
points	Name of a collection of points around which to create the map.	No
route	Name of a route around which to create the map.	No
maneuver	Name of a maneuver around which to create the map.	No

Table 15–10 Map Tag Parameters (Cont.)

Parameter Name	Description	Required
xres	Width of the map in screen display units.	Yes
yres	Height of the map in screen display units.	Yes

The following example of the `map` tag creates a map named `NEDCSmall` 400 pixels wide and 300 pixels high. The center point for the map is the address defined by the `address` tag enclosed in the `map` tag.

```
<loc:map name="NEDCSmall" type="oracle.panama.spatial.jsptags.beans.Map"
       xres="400" yres="300">
   <loc:address
     name="NEDC"
     type="oracle.panama.model.Location"
     businessName="NEDC"
     firstLine="1 Oracle Dr"
     city="Nashua"
     state="NH"
     postalCode="03062"
     country="US"/>
</loc:map>
```

15.2.1.9 route

The `route` tag specifies a route with a specified map resolution. It includes maneuvers, an overview map, and maneuver maps.

Table 15–11 lists the `route` tag parameters. (See Section 15.2.1 for an explanation of the information provided.)

Table 15–11 Route Tag Parameters

Parameter Name	Description	Required
name	Name for the returned object. Example: `myRoute`	Yes
type	Type of object. Must be: `oracle.panama.spatial.jsptags.beans.Route`	Yes
xres	Width of the displayed route in screen display units.	Yes
yres	Height of the displayed route in screen display units.	Yes

The following example of the `route` tag specifies the route between two addresses (an Oracle office in New Hampshire and Oracle headquarters in California).

```
<loc:route name="myRoute" type="oracle.panama.spatial.jsptags.beans.Route"
       xres="800" yres="600">
   <loc:address
        name="NEDC"
        type="oracle.panama.model.Location"
        businessName="NEDC"
        firstLine="1 Oracle Dr"
        city="Nashua"
        state="NH"
        postalCode="03062"
        country="US"/>
   <loc:address
        name="HQ"
        type="oracle.panama.model.Location"
        businessName="HQ"
        firstLine="500 Oracle Parkway"
        city="Redwood City"
        state="CA"
        postalCode="94065"
        country="US"/>
</loc:route>
```

15.2.1.10 JSP Examples for Location Services

This section includes several examples of JSP code to perform operations that involve location services. In these examples, addresses are specified in the `points` attribute of the appropriate tag (`<map>` or `<route>`).

Example 15–4 displays small and large maps of two locations.

Example 15–4 Mapping Using JSP Tags

```
<%@ taglib uri="LocationTags" prefix="loc" %>

<%!
  public String transformString(String orig)
  {
    String result = "";
    for (int i=0;i<orig.length();i++)
    {
      if (orig.charAt(i) == '&')      result = result + "&amp;";
      else if (orig.charAt(i) == '<') result = result + "&lt;";
      else if (orig.charAt(i) == '>') result = result + "&gt;";
```

```
      else                              result = result + orig.charAt(i);
    }
    return result;
  }
%>

<SimpleResult>
    <loc:address
      name="NEDC"
      type="oracle.panama.model.Location"
      businessName="NEDC"
      firstLine="1 Oracle Dr"
      city="Nashua"
      state="NH"
      postalCode="03062"
      country="US"/>
    <loc:map
      name="NEDCSmall" type="oracle.panama.spatial.jsptags.beans.Map" xres="400"
         yres="300" points="NEDC">
    </loc:map>

    <loc:address
      name="HQ"
      type="oracle.panama.model.Location"
      businessName="HQ"
      firstLine="500 Oracle Parkway"
      city="Redwood City"
      state="CA"
      postalCode="94065"
      country="US"/>
    <loc:map name="HQSmall" type="oracle.panama.spatial.jsptags.beans.Map"
       xres="400" yres="300" points="HQ">
    </loc:map>

    <loc:map name="BothSmall" type="oracle.panama.spatial.jsptags.beans.Map"
       xres="400" yres="300" points="NEDC HQ"/>
    <loc:map name="NEDCLarge" type="oracle.panama.spatial.jsptags.beans.Map"
       xres="800" yres="600" points="NEDC"/>
    <loc:map name="HQLarge"   type="oracle.panama.spatial.jsptags.beans.Map"
       xres="800" yres="600" points="HQ"/>
    <loc:map name="BothLarge" type="oracle.panama.spatial.jsptags.beans.Map"
       xres="800" yres="600" points="NEDC HQ"/>

    <SimpleImage target="<%= transformString(NEDCLarge.toString()) %>"
       src="<%= transformString(NEDCSmall.toString()) %>"/>
```

```
    <SimpleImage target="<%= transformString(HQLarge.toString()) %>"
       src="<%= transformString(HQSmall.toString()) %>"/>

    <SimpleImage target="<%= transformString(BothLarge.toString()) %>"
       src="<%= transformString(BothSmall.toString()) %>"/>
</SimpleResult>
```

Example 15–5 displays the route between two locations and the driving directions (maneuvers).

Example 15–5 Routing Using JSP Tags

```
<%@ taglib uri="LocationTags" prefix="loc" %>

<%!
  public String transformString(String orig)
  {
    String result = "";
    for (int i=0;i<orig.length();i++)
    {
      if (orig.charAt(i) == '&')      result = result + "&amp;";
      else if (orig.charAt(i) == '<') result = result + "&lt;";
      else if (orig.charAt(i) == '>') result = result + "&gt;";
      else                            result = result + orig.charAt(i);
    }
    return result;
  }
%>

<SimpleResult>
  <loc:address
    name="NEDC"
    type="oracle.panama.model.Location"
    businessName="NEDC"
    firstLine="1 Oracle Dr"
    city="Nashua"
    state="NH"
    postalCode="03062"
    country="US"/>
  <loc:address
    name="HQ"
    type="oracle.panama.model.Location"
    businessName="HQ"
    firstLine="500 Oracle Parkway"
```

```
    city="Redwood City"
    state="CA"
    postalCode="94065"
    country="US"/>
  <loc:route name="myRoute" type="oracle.panama.spatial.jsptags.beans.Route"
     xres="800" yres="600" points="NEDC HQ">
  </loc:route>

  <SimpleImage src="<%= transformString(myRoute.getMap()) %>"/>

  <SimpleText>
    <loc:iterateManeuvers name="aManeuver"
type="oracle.panama.spatial.jsptags.beans.Maneuver" routeID="myRoute">
      <SimpleTextItem>
        <%= aManeuver.getNarrative() %>
      </SimpleTextItem>
    </loc:iterateManeuvers>
  </SimpleText>
</SimpleResult>
```

Example 15–6 displays business directory (YP) information by name within a specified distance of a location: specifically, a map with the ten Starbucks locations nearest to Oracle headquarters.

Example 15–6 Business Directory (YP) by Name Using JSP Tags

```
<%@ taglib uri="LocationTags" prefix="loc" %>

<%!
  public String transformString(String orig)
  {
    String result = "";
    for (int i=0;i<orig.length();i++)
    {
      if (orig.charAt(i) == '&')      result = result + "&amp;";
      else if (orig.charAt(i) == '<') result = result + "&lt;";
      else if (orig.charAt(i) == '>') result = result + "&gt;";
      else                            result = result + orig.charAt(i);
    }
    return result;
  }
%>

<SimpleResult>
  <loc:address
```

```
    name="HQ"
    type="oracle.panama.model.Location"
    businessName="HQ"
    firstLine="500 Oracle Parkway"
    city="Redwood City"
    state="CA"
    postalCode="94065"
    country="US"/>

  <loc:businesses
    name="starbucks"
    type="java.util.Collection"
    businessName="Starbucks"
    centerID="HQ"
    nearestN="10"/>
  <loc:map name="starbucksMap" type="oracle.panama.spatial.jsptags.beans.Map"
     xres="800" yres="600" points="starbucks">
  </loc:map>

  <SimpleImage src="<%= transformString(starbucksMap.toString()) %>"/>

  <SimpleText>
  <loc:iterateBusinesses name="singleStarbucks" type="oracle.panama.model.Point"
     collection="starbucks">
    <SimpleTextItem> <%= singleStarbucks %> </SimpleTextItem>
  </loc:iterateBusinesses>
  </SimpleText>
</SimpleResult>
```

Example 15–7 displays business directory (YP) information by category within a specified area: specifically, a map with all automobile dealers (new cars) in San Francisco, California.

Example 15–7 Business Directory (YP) by Category Using JSP Tags

```
<%@ taglib uri="LocationTags" prefix="loc" %>

<%!
  public String transformString(String orig)
  {
    String result = "";
    for (int i=0;i<orig.length();i++)
    {
      if (orig.charAt(i) == '&')      result = result + "&amp;";
      else if (orig.charAt(i) == '<') result = result + "&lt;";
```

```
      else if (orig.charAt(i) == '>') result = result + "&gt;";
      else                            result = result + orig.charAt(i);
    }
    return result;
  }
%>

<SimpleResult>
  <loc:category name="automotive"
type="oracle.panama.spatial.yp.YPCategory" categoryName="Automotive">
  </loc:category>

  <loc:category name="automotiveDealers"
     type="oracle.panama.spatial.yp.YPCategory" categoryName="Dealers"
     parentCategory="automotive">
  </loc:category>

  <loc:category name="newAutomotiveDealers"
     type="oracle.panama.spatial.yp.YPCategory" categoryName="New"
     parentCategory="automotiveDealers">
  </loc:category>

  <loc:businesses name="dealers" type="java.util.Collection"
     categoryID="newAutomotiveDealers" country="US" state="CA"
     city="San Francisco"/>

  <loc:map name="dealerMap" type="oracle.panama.spatial.jsptags.beans.Map"
     xres="800" yres="600" points="dealers">
  </loc:map>

  <SimpleImage src="<%= transformString(dealerMap.toString()) %>"/>

  <SimpleText>
  <loc:iterateBusinesses name="dealer" type="oracle.panama.model.Point"
     collection="dealers">
    <SimpleTextItem>
      <%= transformString(dealer.toString()) %>
    </SimpleTextItem>
  </loc:iterateBusinesses>
  </SimpleText>
</SimpleResult>
```

15.2.2 Creating a Location-Based Application Adapter

You can use the location API to write an adapter, which is a Java program that represents the location-based application to its end users.

This section provides information specific to creating an adapter for a location-based application. It does not describe adapter concepts or the general procedure for writing an adapter, because these topics are covered in the *Oracle9iAS Wireless Developer's Guide*.

15.2.2.1 Geocoding

In a geocoding application, the user is asked for an address and the adapter geocodes that address. Such an application can start by constructing a SimpleForm object for the address, as shown in Example 15–8.

Example 15–8 Constructing a SimpleForm Object

```
Element sf = XML.makeElement(result, "SimpleForm");
sf.setAttribute("target", targetString);
result.appendChild(sf);

Element sfi = XML.makeElement(sf, "SimpleFormItem");
sfi.setAttribute("name",  "adrLine1");
sfi.setAttribute("title", "address line 1");
sf.appendChild(sfi);

sfi = XML.makeElement(sf, "SimpleFormItem");
sfi.setAttribute("name",  "city");
sfi.setAttribute("title", "city");
sf.appendChild(sfi);

sfi = XML.makeElement(sf, "SimpleFormItem");
sfi.setAttribute("name",  "state");
sfi.setAttribute("title", "state");
sf.appendChild(sfi);

sfi = XML.makeElement(sf, "SimpleFormItem");
sfi.setAttribute("name",  "postalcode");
sfi.setAttribute("title", "postal code");
sf.appendChild(sfi);

sfi = XML.makeElement(sf, "SimpleFormItem");
sfi.setAttribute("name",  "country");
sfi.setAttribute("title", "country");
```

```
sf.appendChild(sfi);
```

The next time the adapter is invoked (after the user has entered values into the fields), the adapter can access the data, as shown in Example 15–9.

Example 15–9 Accessing Address Data

```
String
   adrLine1   = sr.getInputArguments().getInputValue("adrLine1"),
   city   = sr.getInputArguments().getInputValue("city"),
   state   = sr.getInputArguments().getInputValue("state"),
   postalcode   = sr.getInputArguments().getInputValue("postalcode"),
   country   = sr.getInputArguments().getInputValue("country");
```

Geocoding can be done with a call, as shown in Example 15–10. (Another format of `SpatialManager.createlocation`, not shown in Example 15–10, specifies a point with longitude and latitude coordinates, in which case a `Location` object is created but no geocoding is done.)

Example 15–10 Geocoding the Address

```
Location address =
   SpatialManager.createLocation(
      companyName,
      adrLine1,   // "500 Oracle Parkway"
      null,
      city,   // "Redwood City"
      state,   // "CA"
      postalcode,   // "94065"
      postalCodeExtension,   // null
      country);   // "US"
```

The resulting longitude and latitude values can be accessed as shown in Example 15–11.

Example 15–11 Accessing Values of the Geocoded Address

```
address.getLongitude()
address.getLatitude()
address.getAddressLine1()
address.getCity()
address.getState()
```

Note that the `getLongitude` and `getLatitude` methods are inherited from the `Point` interface.

15.2.2.1.1 International Addresses To better adapt to local address formats, you can use the international address formatting options provided in the `oracle.panama.spatial.intladdress` package. (For information about international address formats, see Section 15.1.3.1.4.) The number of steps necessary to have a user input an address increases by one: the user first has to select a country (address format) in order to be presented with a form for entering the address. Obviously, the form depends on the choice of country, so that the two separate steps cannot be merged to one.

Example 15–12 creates a drop-down `SimpleFormSelect` element that lets the user select an address format (US, German, French, and so on).

Example 15–12 Selecting an Address Format

```
protected void addAdrFormatChoice(
   Element   result,
   String   targetString)
{
   Element sf          = XML.makeElement(result, "SimpleForm");
   sf.setAttribute("target", targetString);
   result.appendChild(sf);
   Element sfs         = XML.makeElement(sf, "SimpleFormSelect");
   sfs.setAttribute("name",  "adrFormat");
   sfs.setAttribute("title", "address format");
   sf.appendChild(sfs);

   Iterator it = IntlAddressManager.getAddressFormats();
   while(it.hasNext())
   {
      String name     = (String)it.next();
      Element sfo     = XML.makeElement(sfs, "SimpleFormOption");
      sfo.setAttribute("value", name);
      sfs.appendChild(sfo);
      sfo.appendChild(XML.makeText(sfo, name));
   }
}
```

The next step is to provide a form requesting all address components relevant to the given address format. The components are determined dynamically based on the chosen country, as shown in Example 15–13.

Example 15–13 Requesting Address Components for a Specified Country

```
protected void chooseAdr(
   Element   result,
   String   targetString,
   String   adrFormat)
{
   Element sf = XML.makeElement(result, "SimpleForm");
   sf.setAttribute("target", targetString);
   result.appendChild(sf);

   Iterator addressComponentNames =
      IntlAddressManager.getAddressFormat(adrFormat).getComponentNames();
   while(addressComponentNames.hasNext())
   {
      Element sfi = XML.makeElement(sf, "SimpleFormItem");
      String name = (String)addressComponentNames.next();
      sfi.setAttribute("name",  "adr_" + name.replace(' ', '_'));
      sfi.setAttribute("title", name);
      sf.appendChild(sfi);
   }
}
```

Example 15–14 displays the result. The components to display and the number of lines depend on the chosen country.

Example 15–14 Displaying Addresses in a Country-Specific Format

```
protected void printResult(
   Element   result,
   String   targetString,
   String   adrComp[],
   String   adrFormat)
{
   IntlAddress loc = IntlAddressManager.createAddress(
      adrFormat,
      adrComp);

   Element sf = XML.makeElement(result, "SimpleText");
   result.appendChild(sf);

   Iterator lines = loc.getAddressLines(false, true);
   while(lines.hasNext())
   {
      Element sfi = XML.makeElement(sf, "SimpleTextItem");
```

```
      sfi.appendChild(XML.makeText(sfi, (String)lines.next()));
      sf.appendChild(sfi);
   }

   Element sfi = XML.makeElement(sf, "SimpleTextItem");
   sfi.appendChild(XML.makeText(sfi, "lat: " + loc.getLatitude()));
   sf.appendChild(sfi);

   sfi = XML.makeElement(sf, "SimpleTextItem");
   sfi.appendChild(XML.makeText(sfi, "lon: " + loc.getLongitude()));
   sf.appendChild(sfi);
}
```

15.2.2.2 Location Marks

An adapter can work with location marks. Example 15–15 retrieves the location marks into an array. (Code not relevant to location marks is omitted from this example.)

Example 15–15 Getting Location Marks

```
public Element invoke (ServiceContext sr)
       throws AdapterException
{
   …

   LocationMark locMarks[] = sr.getSession().getUser().getLocationMarks();

   …
}
```

Note that `LocationMark` extends `Location` (an address).

15.2.2.3 Routing

You can create an adapter that provides routing information between a start address and an end address that the user enters. The adapter must:

1. Set the routing settings and options.
2. Compute the route.
3. Present the resulting route to the user (for example, as a list of maneuvers and maneuver maps, plus an overview map).

Example 15–16 sets the routing settings and options by constructing a `RoutingSettings` object and specifying the resolution (height and width) of the resulting overview and maneuver maps.

Example 15–16 Setting Routing Settings and Options

```
RoutingSettings rS = new RoutingSettings(true, false);
rS.setSecondaryOption(RoutingOption.overviewMapHeight, "600");
rS.setSecondaryOption(RoutingOption.overviewMapWidth, "800");
rS.setSecondaryOption(RoutingOption.maneuverMapHeight, "600");
rS.setSecondaryOption(RoutingOption.maneuverMapWidth, "800");
```

Example 15–17 computes the route, returning a `RoutingResult` object.

Example 15–17 Computing the Route

```
RoutingResult rR =
   SpatialManager.getRouter().computeRoute(
      startLoc,
      endLoc,
      null,   // via points
      rS,   // routing options
      Locale.US);
```

Example 15–18 presents the resulting route to the user, displaying a list of maneuvers and maneuver maps, plus an overview map. (In this example, code specific to the routing API is shown in bold.)

Example 15–18 Presenting the Route to the User

```
Element sm = XML.makeElement(result, "SimpleMenu");
result.appendChild(sm);
Element smi = XML.makeElement(sm, "SimpleMenuItem");
smi.setAttribute("target", rR.getOverviewMapURL()[0].toString());   // first of
                                                  // possibly several overview maps
sm.appendChild(smi);
Text txt2 = XML.makeText(smi, "Map of complete route");
smi.appendChild(txt2);

Maneuver man[] = rR.getManeuvers();
for(int maneuver = 0; maneuver < man.length; maneuver++)
{
    Element sti = XML.makeElement(sm, "SimpleMenuItem");
    sti.setAttribute("target", man[maneuver].getManeuverMapURL().toString());
    sm.appendChild(sti);
```

```
    String str;
    str = man[maneuver].getNarrative();
    Text txt = XML.makeText(sti, str);
    sti.appendChild(txt);
}
```

15.2.2.4 Mapping

In a typical mapping application, the user enters an address and wants to see a map. Example 15–19 gets the map image URL of an address (loc) to be mapped. (The variable loc of type Location contains an address that had been previously geocoded.)

Example 15–19 Getting a Map Image URL:

```
String url =
   SpatialManager.getMapper().getMapURL(
      loc,
      oracle.panama.imagex.ImageFormats.GIF,
      800,   // width
      600,   // height
      false);   // allow turning
```

In Example 15–19, the last parameter specifies whether or not the API can switch the width and height of the image to fit the map better to some mobile device screens. In this example, this option is disabled.

As alternatives to passing a single point object as the first parameter as shown in Example 15–19, you can pass an array of `Point` objects or an object of type `Location` (address) or `YPBusiness`, which extend the `Point` interface.

15.2.2.5 Business Directory (YP)

In a typical business directory (YP) application, the user enters a region specifying a country, state, and city, and wants to get businesses in some category, such as relating to wine tasting or wineries. The user must be asked for country, state, and city, and the application must determine the exact category and then all the relevant businesses.

The first step in determining the category is usually to ask the user for a category keyword (for example, *wine*) through a `SimpleForm` object.

The next step is to determine all the categories that match the keyword, as shown in Example 15–20.

Example 15–20 Finding Categories Matching a Keyword

```
YPFinder ypF = SpatialManager.getYPFinder();
YPCategory cats[] = ypF.getCategoryAtRoot().getCategoriesMatchingName(keyword);
```

Example 15–21 shows a simple user interface that presents categories from which to choose. The user is presented a drop-down menu from which select the category that best matches what he or she is looking for.

Example 15–21 User Interface for Selecting a Category

```
Element sf = XML.makeElement(result, "SimpleForm");
sf.setAttribute("target", targetString);
result.appendChild(sf);
Element sfs = XML.makeElement(sf, "SimpleFormSelect");
sfs.setAttribute("name",  "category");
sfs.setAttribute("title", "Business category");
sf.appendChild(sfs);

for(int i = 0; i < cats.length; i++)
{
   String name = cats[i].getFullyQualifiedName();
   Element sfo = XML.makeElement(sfs, "SimpleFormOption");
   sfo.setAttribute("value", name);
   sfs.appendChild(sfo);
   sfo.appendChild(XML.makeText(sfo, name));
}
```

When the adapter determines the fully qualified name of the chosen category, you can obtain the appropriate category, as shown in Example 15–22.

Example 15–22 Finding the Category

```
YPCategory cat = YPCategory.fromFullyQualifiedName(categoryNameString);
YPBusiness b[] = SpatialManager.getYPFinder().getBusinessesInCity(cat, country,
state, city, Locale.US);
```

The conversion in Example 15–22 from category to `String` back to category is required because a drop-down menu lets you make a selection among `String` objects, not among general objects.

15.2.2.6 Traffic

To create an application based on the traffic services API, you must do the following:

1. Prepare input objects (such as `CityInfo`, `RouteInfo`, `Point`, and `Location`) for the query.
2. Get `TrafficReporter` and summit the query.
3. Obtain `TrafficReport` and process the information.

The rest of this section contains examples of typical operations. Example 15–23 performs a city-level query.

Example 15–23 City-Level Query

```
TrafficReporter reporter = SpatialManager.getTrafficReporter();
CityInfo c = new CityInfo("BOSTON", "MA", "US");
TrafficReport report = null;
try{
  report = reporter.getReportViaCity(c);
}catch(LBSException e){
  System.out.println(e.getLocalizedMessage());
}
```

Example 15–24 performs a route-level query without specifying a direction, and returns incidents in both directions.

Example 15–24 Route-Level Query (Incidents in Both Directions)

```
RouteInfo r = new RouteInfo("US 3", null);
try{
  report = reporter.getReportViaRoute(r,c);
}catch(LBSException e){
  System.out.println(e.getLocalizedMessage());
}
```

Example 15–25 performs a route-level query for a specified direction (north).

Example 15–25 Route-Level Query Specifying Direction

```
try{
  report = reporter.getReportViaRoute(r,TrafficReporter.North,c);
}catch(LBSException e){
  System.out.println(e.getLocalizedMessage());
}
```

Example 15–26 performs a route-level query for an area 10 miles around a specified longitude/latitude point.

Example 15–26 Route-Level Query Around Longitude/Latitude Point

```
p = SpatialManager.createPoint(-71.0607, 42.3659);
try{
  report = reporter.getReportViaLocation(p, 10, TrafficReporter.MILES,
c);
}catch(LBSException e){
  System.out.println(e.getLocalizedMessage());
}
```

Example 15–27 performs a route-level query for an area 10 miles around a specified address.

Example 15–27 Route-Level Query Around Address

```
Location loc = SpatialManager.createLocation(null, null, "839 Kearny
     Street", null, "San Francisco", "CA", null, null, "US");
try{
  report = reporter.getReportViaAddress(loc, 10, TrafficReporter.MILES);
}catch(LBSException e){
  System.out.println(e.getLocalizedMessage());
}
```

Example 15–28 processes a traffic report to get useful information.

Example 15–28 Processing a Traffic Report

```
Calendar rTime = report.getReportTime();
TrafficIncident[] incidents = report.getIncidents();
if(incidents != null){
  for(int i=0; i<incidents.length; i++){
    TrafficIncident inc = incidents[i];
    String desc = inc.getDescription();
    String severity = inc.getSeverity();
    String type = inc.getType();
    TrafficRoute route = inc.getIncidentRoute();
    String[] locations = inc.getLocationRange();        //text description
    if(locations.length == 2){                          //a location range
      String exit1 = locations[0];
      String exit2 = locations[1];
    }
    else if(locations.length == 1){
```

```
      String exit1 = locations[0];                              //one location
    }
    Point geoLocation = inc.getIncidentLocation();      //lon/lat or
lon/lat+radius
    Calendar[] tr = inc.getTimeRange();
  }
}
```

Example 15–29 returns a list of cities for which traffic support is provided.

Example 15–29 Returning a List of Cities

```
TrafficCityManager manager = reporter.getCityManager();
CityInfo[] cities = null;
try{
  cities = manager.getActiveCities();
}catch(LBSException e){
  System.out.println(e.getLocalizedMessage());
}
```

Example 15–30 returns a list of routes for which traffic support is provided in a specified city (San Francisco, California).

Example 15–30 Returning a List of Routes in a City

```
TrafficCityManager manager = reporter.getCityManager();
CityInfo sf = new CityInfo("SAN FRANCISCO", "CA", "US");
RouteInfo[] routes = null;
try{
  routes = manager.getRoutesInCity(sf);
}catch(LBSException e){
  System.out.println(e.getLocalizedMessage());
}
```

15.3 Enabling Mobile Positioning

You can enable mobile positioning for individual users or groups of users of a location-based application. **Mobile positioning** of a user refers to associating a location with that user. When mobile positioning is enabled for a user, the user's current location, whether it is obtained dynamically from automatic positioning or from a default location mark, is used by Oracle9*i*AS Wireless to determine the visibility of location-based services or folders. A service or folder can be defined as location-dependent (as described in Section 15.4.4) by associating it with a system region or a previously defined custom region. A location-dependent service or

folder appears in a user's portal only when the user's current location (from automatic positioning or from a default location mark) is within the associated region. For example, if the user's current location is in Boston, a Boston traffic information service would be visible to the user; otherwise, the service would not be visible to that user.

Mobile positioning can be manual or automatic:

- **Manual positioning** occurs when a specific location is assigned to a user. The assigned location could be the geocoded result of an address that the user is asked to enter, an explicitly specified *location mark*, or a default location for the user. For example, the location of the user's home might be specified for mobile positioning, and an application could then offer information and options relevant to that home area (regardless of the user's actual current physical location).
- **Automatic positioning** (sometimes called *location acquisition*) occurs when the user's location is determined automatically based on positioning information based on the location of the mobile device. For example, the location of a delivery truck driver or service technician might be periodically determined based on the person's mobile device location, and an application could consider that location data when providing information or instructions to the user.

  Automatic positioning provides several options relating to frequency of position updates and user privacy.

This section describes manual and automatic positioning in more detail, and describes how to enable each type of positioning.

15.3.1 Manual Positioning

Manual positioning associates a specific location with the mobile application user. The location can be explicitly specified (such as the user entering an address or the name of a location mark), or it can be a default location mark for that user. A location mark is a position that is typically associated with longitude and latitude coordinates and that has a name. For example, an application user can create location marks named `MyHome` and `MyOffice` (for the person's home and office locations, respectively), and associate a geocoded address with each one. If this person designated `MyHome` as the default location mark, the mobile application would consider the person's home address as the person's location.

If a user tries to set a default location mark that is not geocoded, a geocoding operation is performed before the location mark is made the default. If the geocoding operation fails, it is recommended that you not set that location mark as

the default, because many capabilities (such as location-dependent service visibility) depend on the geocoded information of the default location mark.

For more information about location marks, see Section 15.1.5.

15.3.1.1 Enabling Manual Positioning

To enable manual positioning for a user, first set up any location marks that you might want to use. Use the API (`LocationMark` class) or the Personalization Portal Web interface to create one of more location marks (if they do not already exist), and specify a location mark as the default for that user.

> **Note:** If automatic positioning (described in Section 15.3.2) is turned off or if the positioning server is temporarily unavailable, manual positioning is used, and the user's default location mark is used. (Automatic positioning can be turned on and off using the Oracle9*i*AS Wireless webtool.)

To enable manual positioning using the Personalization Portal interface, follow these steps:

1. Log in to the Personalization Portal Web interface
2. Click the **LocationMarks** tab.
3. If the location mark that you want for your default location does not already exist it, create it. (Click **Create** and complete the information on the page that is displayed.)
4. Select the location mark that you want for your default location.
5. Click **Set Default**.

15.3.2 Automatic Positioning

Automatic positioning allows the user's location to be determined based on a position based on the location of the user's mobile device. You can determine how timely, and thus potentially how accurate, the location is by setting a positioning quality of service (QoS) value.

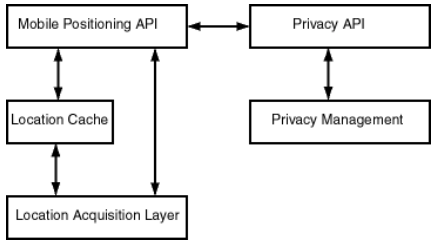
The Oracle9*i*AS Wireless API enables an application to access a mobile user's current location through the current session (see `getCurrentLocation()` in the `oracle.panama.rt.Session` interface). If automatic positioning is turned on in the system, the user's current physical location is returned from the mobile

positioning system. If automatic positioning is turned off or if the positioning server is temporarily unavailable, the user's default location mark is returned.

Privacy and the security of privacy-related information are important concerns in a location acquisition system. The Oracle9*i*AS Wireless location services provide a privacy management component that allows users to view and edit their privacy settings, to enable and disable the positioning operation on themselves, and to authorize one or more people (a mobile community) to obtain positioning information on them within certain time frames. It also allows application developers to access these capabilities through a public API.

Automatic positioning is controlled by the mobile positioning framework, which is shown in Figure 15–7.

Figure 15–7 Mobile Positioning Framework



As Figure 15–7 shows:

- Application developers can use the mobile positioning API together with the privacy API to provide services.
- The mobile positioning API in the application communicates with the location cache (described in Section 15.3.2.1) and the location acquisition layer to determine the user's location. Whether or not the cache is used is affected by the positioning quality of service (QoS) value, which is described in Section 15.3.2.2.
- The location acquisition layer passes the actual current position to the location cache and to the mobile positioning API.
- Privacy management logic controls privacy-related aspects of the mobile positioning framework, which are described in later sections.

15.3.2.1 Location Cache

The location cache is an area in memory that temporarily stores a mobile user's ID, the most recently acquired location information, and the time when that information was gathered. If the location cache is searched for a mobile positioning request, and if there is an entry in the cache for the user whose location is requested, the time difference between the cache entry time and the current request time is compared against the positioning quality of service level of the positioning request. (Positioning quality of service is explained in Section 15.3.2.2.)

When a positioning request is satisfied by information in the cache, no position sensing is required; that is, no network round-trip operation is required.

15.3.2.2 Positioning Quality of Service

The positioning quality of service (QoS) value controls:

- Whether to check the current device position or the location cache to determine the location.
- If the location cache is checked, a maximum "age" of the most recent cached location value (that is, a number of seconds since that value was written to the location cache) for it to be used by the application.

You can specify the positioning quality of service value in either of the following ways:

- As a number of seconds, representing the maximum age of the position in the location cache for it to be used by the application. If the most recent position in the location cache is older than the appropriate time, the actual current position of the device is obtained, written in the cache, and used by the application. A value of 0 (zero) causes the positioning framework always to give the actual positioning result and not to search the location cache.
- As one of the following string values, each representing a level of positioning quality:
  - Exact: Causes the positioning framework always to give the actual positioning result and not to search the location cache; equivalent to specifying 0 (zero) seconds.
  - High: Represents a high level of probable accuracy.
  - Medium: Represents a medium level of probable accuracy.
  - Low: Represents a lower level of probable accuracy than the Medium value.

For the High, Medium, and Low values, the positioning framework determines an age value (number of seconds) in a heuristic manner.

There is a system default positioning quality of service level, which you can set. If a positioning quality of service level is not specified with a positioning request, the system default is used. The level can be set using the mobile positioning API (see Section 15.3.2.7).

The trade-off in selecting a positioning quality of service level is probable accuracy versus application performance. A value of 0 seconds or Exact guarantees that the actual current position is obtained; however, obtaining the actual position requires network round-trip to the service provider for each mobile positioning request. Such round-trip operations can slow application performance, especially if there are positioning requests for many users or many requests for the same user. You should use a value of 0 seconds or Exact only if the application always needs to know the actual position. A value of Low returns a location that is least likely to be accurate (unless the user has not moved at all); however, it increases the probability that the location will be obtained from the cache, eliminating the need for a network round-trip operation. If the user is not likely to move very far or fast, or if it is not important to know the actual current location, a value of Low may be best.

15.3.2.3 Specifying Positioning Providers

Automatic mobile position is queried by calling the `Positioner.requestPosition` function. (`Positioner` is a class in the `oracle.panama.mp` package). A `Positioner` object is based on one or more mobile positioning providers. As with other location service providers, a mobile positioning is configured by specifying information such as the name, version, URL, user name, and password.

However, a mobile positioning service differs from other location services in that in some cases positioning can only be handled by one specific provider, which is less likely to be true for other location based services. For example, if you request a map of California, several mapping providers are able to provide the map. However, if you request mobile position for a specific phone number (such as +4412345678), it is very likely only one provider can provide the position. A mobile ID (typically a phone number) usually identifies a wireless carrier and thus also determines the mobile positioning provider or providers. Therefore, application developers need to be able to get a positioner based on specific mobile positioning providers.

To meet different application needs, several `getPositioner` signatures are provided in the `MPManager` class:

- `getPositioner()`

- `getPositioner(MPProvider provider)`
- `getPositioner(MPProvider[] providers)`

An Internet portal may have subscribers from different carriers, and they may need to decide dynamically, based on the mobile ID, which provider to use at run time. This need is supported by mobile positioning provider selector hooks (implemented through the `oracle.panama.mp.MPProviderSelector` interface).

A provider **selector hook** takes a mobile ID and returns an array of `MPProvider` objects that can handle the positioning request. The default provider selector hook is provided by `oracle.panama.mp.core.ProviderSelectorImpl`, which returns all providers in the system, which means by default the positioning framework does not attempt to choose a provider. A selector hook is used by a positioner when calling `positioner.requestPosition` and is applicable for the `getPositioner()` signature. If the providers are already specified when the positioner is called, the selector hook is not used.

The Oracle9*i*AS Wireless API enables an application to access a mobile user's current location through the current session (`Session.getCurrentLocation()`). By default, the user's mobile ID (which is in the user's profile) is used to call the mobile positioning API to get the current position. However, if advanced users want to use a different value for positioning, they can write their own mobile ID hook by implementing the `oracle.panama.rt.hook.MobileIDHook` interface. A mobile ID hook takes the current user's information and returns his or her mobile ID for positioning. If the automatic positioning fails, the system fails over to the user's default location mark as the current location.

Note that you do not have to implement either the provider selector hook or the mobile ID hook. If the default settings meet your needs, you can simply configure mobile positioning providers and call `MPManager.getPositioner().requestPosition()`.

To summarize:

- A `Positioner` can be a system default based on all mobile positioning providers configured in the system, or it can be customized based only on one or more specific providers.
- When a system default is used, a provider selector hook is used only when choosing the system default positioner. A selector hook takes a mobile ID and decides which provider or providers can handle it. In the case of batch query, the first mobile ID in the batch determines which provider is selected.

- Failover is provided when a positioner is based on more than one provider and a provider cannot handle the request.

Programs should check that the `PositionResult` has a nonzero error code before using it.

Example 15–31 gets the user's position using system default providers and the default positioning quality of service.

Example 15–31 Getting Position using System Default Providers and Default QoS

```
Positioner positioner = MPManager.getPositioner();
PositionResult res = positioner.requestPosition("46708123456790");
Date timeStamp = res.getTimeStamp();
double lon = res.getPositionAreas()[0].getCenterPointLongitude();
double lat = res.getPositionAreas()[0].getCenterPointLatitude();
```

Example 15–32 shows two examples of getting the user's position and specifying a positioning quality of service level. The first example specifies the quality descriptively as high, and the second example specifies the quality as a number of seconds. (Section 15.3.2.2 explains the ways in which you can specify positioning quality of service.)

Example 15–32 Getting Position Specifying QoS

```
PositionResult res = positioner.requestPosition("46708123456790",
ServiceQoS.HIGH_QUAL);

PositionResult res = positioner.requestPosition("46708123456790", new
ServiceQos(120));
```

Example 15–33 gets the user's position based on an array of specific providers.

Example 15–33 Getting Position Based on an Array of Providers

```
MPProvider[] providers = new MPProvider[2];
providers[0] = MPManager.lookup("CellPoint", "1.2");
providers[1] = MPManager.lookup("Ericsson", "3.0");
Positioner positioner = MPManager.getPositioner(providers);
```

15.3.2.4 Granting and Revoking Positioning Rights

By default a user's location information can only be accessed by himself or herself. No other user has the right to access the user's location information. If users want to allow other users to access their location information, they must grant the

positioning right to those users. A user granting the positioning right can later revoke the granted right.

The positioning right can also be granted for a certain duration or recurring interval of time. In many cases, users want to restrict the time periods to grant other users the right to access their location information. For example, users may want to grant coworkers this right from 9:00 am to 5:00 pm during weekdays, but they do not want coworkers to position them at night or during weekends. Users can specify such time restrictions as:

- Starting and ending dates of the granted right
- Starting and ending time during a day
- Exclusions: days that are within the start and end dates but are excluded from the positioning right, such as Saturdays and Sundays

15.3.2.5 Mobile Communities

A **mobile community** is a collection of one or more users who can be granted or denied positioning rights. Mobile users can be assigned to one or more communities, and users can grant and deny positioning rights to communities. Users can view and manage their community information through the Personalization Portal, and application developers can access these capabilities through the public API.

The concept of mobile community is useful in many mobile application scenarios. For example, a project team can create a project community. A team member can grant to the project community the right of accessing to his or her location information instead of granting the right to each team member individually. For example, with mobile positioning and location-based alerts, a field service manager could know when service representatives are nearby and could contact them to get status updates or to have them respond to local problems.

The concept of **visibility** applies to communities and to members of communities. Visibility refers to the ability of system users to see that a community or member exists and to obtain some relevant information. Visibility can depend on the relationship of the requesting user to the community or member: for example, whether the requesting user has administrator privileges or is a member of the community in question. Visibility is implemented using calls to the privacy API, which is described in Section 15.3.2.8.

For any given request by a user to see information about a community or members of a community, the following visibility conditions are possible:

- The community and the members of the community are visible to the requesting user.
- The community is visible to the requesting user, but the members of the community are not visible. For example, the community has been set up so that its existence is visible to all system users; however, information about community members is available only to administrators.
- The community is not visible to the requesting user, and therefore members of the community are not visible either.

Different types of communities are supported, to accommodate different user requirements for visibility. When you create a community, you can specify the type of community, namely:

- Private: A private community is visible only to the creator of the community, who has sole and complete control. No other users, including members of the community, can see or perform operations on a private community.
- Shared: A shared community is visible to all the community members but not to other users in the system. A community member is visible to all other community members. A community member can remove himself or herself from the community.
- Public with Member Visibility: A public community with member visibility is visible to all the users in the system. Any users in the system can add themselves to the community and remove themselves from the community.
- Public Member-Controlled Visibility: A public community with member-controlled visibility is visible to all the users in the system; however, each member can control whether he or she is visible or not visible to other users.
- System: A system community is visible to all users of the system, but the members are visible only to users who have administrator privileges. Users without administrator privileges cannot remove themselves from a system community.

The following community operations are supported:

- Create a community and add initial members
- Delete a community
- View a list of all the communities that are visible to the user
- View all the members in the community who are visible to the user

- Add users to a community (for the creator of a community)
- Remove users from a community (for the creator of a community, or any community member for removing himself or herself from a shared community)

15.3.2.6 Privacy Directives and Enabling or Disabling Automatic Positioning

With the initial default privacy settings, the system does not have the right to position a user and temporarily store the user's position in the location cache, and write the user's location information to the cache log. However, the administrator can specify a different system default level of privacy -- and users can control their level or privacy through the Personalization Portal -- by using any of the following privacy directives, listed in decreasing order of privacy provided:

- Disable Positioning and Caching: No positioning on the user is allowed. The system has no right to position the user, and no access to the user's location is allowed. This setting provides the most privacy.
- Enable Positioning, Disable Caching: The user's location information is not cached. The system has the right to position the user, but the system cannot store the user's location information in the location cache. In this case, the user's location is always obtained by going to the positioning service providers directly.

  For example, with this directive a mobile user's movements might not be tracked, and the position at any time might be reported as the user's office or whatever location the service provider supplies.
- No Log: The user's location information is stored in the location cache, but is not written to the cache log. Cache items for this user are not written to the log when they are replaced from the cache, but are simply discarded.

  For example, with the No Log directive, a mobile user's current position might be available, but earlier positions might not be available if they had discarded from the location cache.
- Enable Positioning and Caching: The system has the right to acquire and cache the user's location information.

15.3.2.7 Mobile Positioning API

Mobile device positioning is performed by calling the corresponding `requestPosition` functions in the `Positioner` class. The API allows application developers to specify the positioning quality of service (QoS) level. (These levels are explained in Section 15.3.2.2.)

15.3.2.8 Privacy API

Developers of mobile applications can manage the privacy capabilities through the location services privacy API. This section describes the privacy API and provides examples.

15.3.2.8.1 LocationPrivacyManager Class The `LocationPrivacyManager` class handles all the location privacy-related operations, such as granting and revoking positioning rights, enabling and disabling positioning rights, setting and getting system privacy options, and checking if a user has right to position another user. The class also provides ways to retrieve the `LocationPrivacyAuth` object, which stores information about a privacy authorization item.

A user can grant authorization to another user or to a mobile community using `grantAuthorization`. The authorization can be temporarily disabled using `disableAuthorization`. The disabled authorization can be recovered by using `enableAuthorization`. The granted right can be permanently revoked using `revokeAuthorization`. `checkAuthorization` can be used to check whether a user has right to position another user at specific time.

All the privacy authorization operations are application-specific, which means that they only affects the application in which the operation is performed.

15.3.2.8.2 CommunityManager Class The CommunityManager class handles community-related operations, such as creating and deleting community and retrieving community information. Community operations specific to a single community are defined in the `Community` interface.

15.3.2.8.3 LocationPrivacyAuth Interface The `LocationPrivacyAuth` interface provides methods to retrieve information specific to a location authorization item, such as the authorization period, the service where the authorization occurs, the user granting the right, and the user receiving the right.

15.3.2.8.4 Community Interface The `Community` interface provides methods to retrieve information about the community object, add members to and remove members from the community, and set community attributes.

15.3.2.8.5 AuthPeriod Class The `AuthPeriod` class maintains a time range that is used when a user grants the positioning right to other users. An authorization period is composed of start date, end date, start time, end time, and exclusions. The class also provides a method contains to check whether a specific time falls in the time range represented by the class.

15.3.2.8.6 LocationPrivacyException Class The `LocationPrivacyException` class is a subclass of `PanamaException`. It represents a location privacy-specific exception.

15.3.2.8.7 Privacy API Examples This section contains examples of the location services privacy API. The examples are taken from the sample adapters SampleCommunityManager.java and SampleFriendFinder.java is the iAS-wireless-home\sample\sampleadapter\mp\privacy directory. These two sample adapters demonstrate the major capabilities of the privacy API.

Example 15–34 lists all communities of a specified type that are visible to a user.

Example 15–34 List Communities of a Specified Type Visible to a User

```
CommunityManager commMan = CommunityManager.getInstance();
...
try{
     ResultSetEnumeration comms = commMan.getVisibleCommunities(user,type);
     while (comms.hasNextElement()){
          sfo = XML.makeElement(sfs,"SimpleFormOption");
          oracle.panama.model.Community comm =
(oracle.panama.model.Community)(comms.next());
          sfo.setAttribute("value",String.valueOf(comm.getId()));
         txt = XML.makeText(sfo,comm.getCreator().getName()+":"+ comm.getName()
);
          sfo.appendChild(txt);
          sfs.appendChild(sfo);
    }
  }catch(Exception e){ throw new AdapterException(e);    }
```

Example 15–35 grants the positioning right to a user or a community based on user input.

Example 15–35 Grant Positioning Right to a User or Community

```
CommunityManager commMan = CommunityManager.getInstance();
LocationPrivacyManager priMan = LocationPrivacyManager.getInstance();

...

SimpleDateFormat ddf = new SimpleDateFormat("MM/dd/yyyy");
SimpleDateFormat tdf = new SimpleDateFormat("HH:mm");
Calendar startD,endD,startT,endT=null;
try{
    startD = Calendar.getInstance();
```

```
    startD.setTime(ddf.parse(sdate));

    endD = Calendar.getInstance();
    endD.setTime(ddf.parse(edate));

    startT = Calendar.getInstance();
    startT.setTime(tdf.parse(stime));

    endT = Calendar.getInstance();
    endT.setTime(tdf.parse(etime));

}catch(ParseException e){
      showError(result,sr,"Illegal Date Format","&grantmenu=y");
      return;
}

StringTokenizer st = new StringTokenizer(excl,",");
String exclDate = null;
byte exclusions=0;
while (st.hasMoreTokens()) {
    exclDate=st.nextToken();
    if ("Mon".equals(exclDate))
         exclusions =(byte)(exclusions|AuthPeriod.MONDAY);
    else if ("Tue".equals(exclDate))
         exclusions =(byte)(exclusions | AuthPeriod.TUESDAY);
    else if ("Wed".equals(exclDate))
         exclusions =(byte)(exclusions | AuthPeriod.WEDNESDAY);
    else if ("Thu".equals(exclDate))
         exclusions =(byte)(exclusions | AuthPeriod.THURSDAY);
    else if ("Fri".equals(exclDate))
         exclusions =(byte)(exclusions | AuthPeriod.FRIDAY);
    else if ("Sat".equals(exclDate))
         exclusions =(byte)(exclusions | AuthPeriod.SATURDAY);
    else if ("Sun".equals(exclDate))
         exclusions =(byte)(exclusions | AuthPeriod.SUNDAY);
    else {
        showError(result,sr,"Illegal Exclusions.", "&grantmenu=y");
        return;
    }
 }

 AuthPeriod period = new AuthPeriod(startD,endD, startT,endT, exclusions);
 oracle.panama.model.Community commObj = null;
 User posUserObj = null;
 try{
```

```
    if (community!=null && !community.equals("")){
       commObj = commMan.getCommunity(Long.parseLong(community));
       priMan.grantAuthorization(service,owner,commObj,period);
    }
   else{
      posUserObj = services.lookupUser(positionUser);
      priMan.grantAuthorization(service,owner,posUserObj,period);
  }
}catch(PanamaException e){ throw new AdapterException(e);   }
```

15.4 Using the Region Modeling Tool

The region modeling tool lets administrators of a wireless portal service manage regions and make a service or folder location-dependent. When you create a service or a folder, you can specify that it is location-dependent by associating a system region or a previously created custom region with the service or folder. A location-dependent service or folder appears is a user's portal only when the user's current location (either from automatic mobile positioning or from the user's default location mark) is within the specified region.

A **region** is simply a geographic entity, or location. A region can be small (such as a street address) or large (such as a country). A region can be represented by a point, as is often done for addresses and locations of interest (such as airports and museums), or by a polygon, as is usually done for states and countries.

15.4.1 Service and Folder Visibility Using Region Modeling

You may want to define specific regions for a variety of applications and services, such as:

- City guides for selected metropolitan areas, so that users in those areas receive only services and information (such as restaurant listings or advertisements) relevant to them
- Colleges that have a certain ranking or that specialize in certain subject areas, so that prospective students and their parents can receive information about those locations
- Art museums in a city or a multistate area, so that art lovers can plan trips to museums

Your company may provide many specialized services, and users may be able to subscribe to and pay for individual services tied to regions. For example, one user

might subscribe to city guides for the entire United States, while another user might subscribe only to city guides for southeastern states.

To implement the city guide example, you could do the following:

1. Create a folder (static, not location dependent) called *City_guide*.
2. Under the City_guide folder create city guide services for Boston, San Francisco, and California
3. Set the default location mark to an address in a city. If the address is in Boston, the user sees the Boston city guide; if the address is in San Francisco, the user sees both the San Francisco and California guides.

In another example scenario, several services may be relevant to a region, in which case you can create a location-dependent folder and place the relevant services in that folder (instead of designating each service as location-dependent on the region). For example, assume that you have ATM Locator, Flight Gate Information, Airport Parking Information, and Taxi Finder services associated with a region named Airport, and that you have Printer Finder, Conference Room Scheduler, and Cafeteria Menu services associated with a region named Office. In this case, you can create two location-dependent folders named Airport and Office, and associate them with the Airport and Office regions, respectively.

15.4.2 Folders and Hierarchies of Regions

Regions are stored in folders. Folders can be in a hierarchy (that is, there can be folders in folders). There are two top-level folders: System-Defined Regions and Custom Regions.

- **System-defined regions** are arranged in a hierarchy of predefined areas: continents, which contain countries. The United States further contains states, which contain postal codes, counties, and cities.
- **Custom regions** are regions created by users, based on entering an address or on selecting one or more other regions (system-defined or custom).

15.4.3 Region Modeling Tool Web Interface

The region modeling tool's Web interface is part of the Oracle9*i*AS Wireless Service Designer, which is part of the webtool. In the **Service Designer**, click the **Regions** tab to display the region modeling tool, shown in Figure 15–8.

Figure 15–8 Region Modeling Tool Web Interface

The Web browser window initially displays the top level of the region hierarchy, with two entries: system-defined regions and custom regions. You can find regions, and you can select regions for viewing or for adding to a collection from which you create a custom region.

To find a region, enter a character string that is in its name to search by name, or enter a number to search by ID (by region ID). Specify to search all regions or only under current region (the currently selected region), then click **Go**.

To select a region to perform an operation on it, click the box to the left of its icon and name. (To deselect a selected item, click the box.) You can select all regions in the current display by clicking **Select All**, and deselect all regions in the current display by clicking **Select None**.

To perform an operation on the selected region or regions, click the command text link or button shown in Table 15–12.

Table 15–12 Region Modeling Tool Operations

To Do This:	Click This:
Add selected regions to the collection of regions at the bottom of the display	**Add to Collection**
View a map display showing selected regions	**View**
Create a custom region from the collection of regions at the bottom of the display	**Create from Collection**. A series of pages is then displayed, in which you specify the location in the region hierarchy and the name for the custom region.
Create a custom region from a street address that you enter	**Create from Address**. A series of pages is then displayed, in which you specify the address, the location in the region hierarchy, and the name for the custom region.
Create a custom region from the collection of regions at the bottom of the display	**Create from Collection**. A series of pages is then displayed, in which you specify a location in the region hierarchy and a name for the custom region.
Create a folder in which to organize regions	**Create Folder**. A series of pages is then displayed, in which you specify the location in the region hierarchy under which to create the folder and the name for the folder.
Go to the previous or next set of entries in the display of regions or the current collection	**Previous** or **Next**
Go up one or more levels in the region hierarchy	The name of the desired level on the current hierarchy line near the top of the page. Example of how this line might look (with all items except the last as links): Regions > System Defined Regions > NORTH AMERICA > USA > California
Get help about any screen	**Help**

15.4.4 Associating a Region with a Service

When you create a service and make it location dependent, you must specify the region for which the service applies or is relevant. Before you can specify the region, it must already exist, either as a system-defined region or a custom region. If it is a

custom region, it must have been created using the region modeling tool, using one of the methods described in Section 15.4.3 for creating a region.

To associate a region with a service, you must specify **Location Dependent** when creating the service, and you must click the flashlight icon to select the region. Figure Figure 15–9 shows an example of a service specified as location dependent.

Figure 15–9 Specifying a Service as Location Dependent

When you click the flashlight icon, a window is displayed showing the top of the region hierarchy. You can select any region (but only one region) in the region hierarchy.

To associate the selected region with the service, click **Done**.

15.4.5 Loading and Updating Region Data

The region modeling tool is installed with an extensive set of data for the United States, as well as country data for many countries. However, you can add data about other countries, states, cities, and so on by adding rows to the tables where the region data is stored. For example, you could add a row for each state in India to the STATE table. If you are careful and know what you are doing, you can also

modify certain data in those tables, such as editing the DESCRIPTION column values for certain cities or states.

15.4.5.1 Tables for Region Data

Region data is stored in the Oracle9*i*AS Wireless repository in the tables listed in Table 15–13.

Table 15–13 Tables for Region Data

Table Name	Contains Information About
CONTINENT	Continents
COUNTRY	Countries
STATE	States
COUNTY	Counties
CITY	Cities
POSTALCODE	Postal codes
USERDEFINED	Custom regions

To see the definition of any of these tables, use the SQL statement DESCRIBE. Example 15–36 shows the DESCRIBE statement output with information about all of the tables.

Example 15–36 Region Data Table Definitions

```
SQL> DESCRIBE continent;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL NUMBER
 NAME                                               VARCHAR2(100)
 REFCNT                                             NUMBER
 DESCRIPTION                                        VARCHAR2(2000)
 GEOMETRY                                           MDSYS.SDO_GEOMETRY

SQL> DESCRIBE country;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL NUMBER
 NAME                                               VARCHAR2(300)
 REFCNT                                             NUMBER
 CONT_ID                                            NUMBER
```

```
 DESCRIPTION                                        VARCHAR2(2000)
 GEOMETRY                                           MDSYS.SDO_GEOMETRY

SQL> DESCRIBE state;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL NUMBER
 NAME                                               VARCHAR2(400)
 REFCNT                                             NUMBER
 ABBR                                               VARCHAR2(32)
 CONT_ID                                            NUMBER
 COUNTRY_ID                                         NUMBER
 DESCRIPTION                                        VARCHAR2(2000)
 GEOMETRY                                           MDSYS.SDO_GEOMETRY

SQL> DESCRIBE county;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL NUMBER
 NAME                                               VARCHAR2(400)
 REFCNT                                             NUMBER
 CONT_ID                                            NUMBER
 COUNTRY_ID                                         NUMBER
 STATE_ID                                           NUMBER
 DESCRIPTION                                        VARCHAR2(2000)
 GEOMETRY                                           MDSYS.SDO_GEOMETRY

SQL> DESCRIBE city;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL NUMBER
 NAME                                               VARCHAR2(400)
 REFCNT                                             NUMBER
 CONT_ID                                            NUMBER
 COUNTRY_ID                                         NUMBER
 STATE_ID                                           NUMBER
 DESCRIPTION                                        VARCHAR2(2000)
 GEOMETRY                                           MDSYS.SDO_GEOMETRY
 MIN_LON                                            NUMBER
 MIN_LAT                                            NUMBER
 MAX_LON                                            NUMBER
 MAX_LAT                                            NUMBER

SQL> DESCRIBE postalcode;
 Name                                      Null?    Type
```

```
 ---------------------------------------- -------- ----------------------------
 ID                                       NOT NULL NUMBER
 NAME                                              VARCHAR2(400)
 REFCNT                                            NUMBER
 CONT_ID                                           NUMBER
 COUNTRY_ID                                        NUMBER
 STATE_ID                                          NUMBER
 DESCRIPTION                                       VARCHAR2(2000)
 GEOMETRY                                          MDSYS.SDO_GEOMETRY

SQL> DESCRIBE userdefined;
 Name                                     Null?    Type
 ---------------------------------------- -------- ----------------------------
 ID                                       NOT NULL NUMBER
 NAME                                              VARCHAR2(200)
 REFCNT                                            NUMBER
 TYPE                                              NUMBER
 PARENT_FOLDER_ID                                  NUMBER
 DESCRIPTION                                       VARCHAR2(2000)
 GEOMETRY                                          MDSYS.SDO_GEOMETRY
```

15.4.5.2 Inserting Data into Region Tables

You can use the SQL statement INSERT to insert rows into the region tables. The following considerations apply when you are inserting region data:

- You should use `idseq.nextval` to generate the ID column value whenever you insert a new row, as shown in Example 15–37. The *idseq* sequence is created automatically during the *i*AS installation; you should not create it.
- The REFCNT column should be set to 0 (zero) when you insert a row. The REFCNT column contains the reference count of how many services are associated with the region. The value is automatically incremented when a service is associated with the region and decremented when it is disassociated from the region or when the service is deleted. A region with a nonzero REFCNT value cannot be deleted.
- If you are inserting data about postal codes, cities, or counties for a country that does not use the default region hierarchy, specify 0 (zero) as the STATE_ID in POSTALCODE, CITY, or COUNTY table.
- The GEOMETRY column value must be a valid Oracle Spatial geometry of type MDSYS.SDO_GEOMETRY. The SDO_GTYPE value must be 4 digits, and the SRID (coordinate system) value must be 8307 (for WGS-84 longitude/latitude format). If the SRID value is not currently 8307, you must transform geometries

into that format before inserting them into the region data tables. For detailed information about the spatial data type, coordinate systems, and coordinate system transformation, see the *Oracle Spatial User's Guide and Reference*.

Example 15–37 shows an INSERT statement to insert a row for Concord, Massachusetts into the CITY table. It assumes that the geometry representing Concord exists in another table named MY_CITIES.

Example 15–37 Inserting a City

```
DECLARE
  city_geom MDSYS.SDO_GEOMETRY;

BEGIN

-- Populate geometry variable with city geometry from another table.
SELECT m.geometry into city_geom FROM my_cities m
  WHERE m.name = 'Concord';

-- Insert into the CITY table.
INSERT INTO CITY VALUES (
   idseq.nextval,
   'Concord',
   0,
   5004, -- continent ID for North America
   5006, -- country ID for USA
   5028, -- state ID for Massachusetts
   'The historic town of Concord',
   city_geom,
   -71.37, -- minimum longitude
   42.46, -- minimum latitude
   -71.36, -- maximum longitude
   42.47); -- maximum latitude
END;
/
```

The minimum and maximum longitude and latitude values in the CITY table are required and are used by traffic support services. The minimum longitude and latitude values identify the lower-left corner, and the maximum longitude and latitude values identify the upper-right corner, of the bounding rectangle.

15.4.6 Region Modeling API

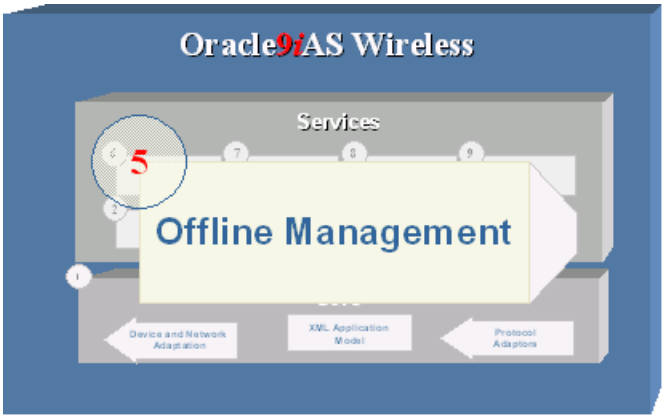
The region modeling tool is based on the region modeling API, which is implemented through the `RegionModel` interface of the `oracle.panama.spatial.region` package. The `RegionModel` interface includes methods for getting the postal code, state, and country, and for determining different kinds of interaction among regions.

16

Offline Management

This document explains Offline Management using Oracle9*i*Lite.

Figure 16–1 Offline Management



16.1 Oracle9*i* Lite: The Internet Platform for Mobile Computing

Oracle9*i* Lite provides infrastructure and application services that enable the delivery of secure and personalized applications using a broad range of mobile devices. Oracle9*i* Lite is an add-on to Oracle9*i*AS, and complements Oracle9*i*AS Wireless, providing a complete, integrated, therefore simple mobile e-business framework. Oracle9*i* Lite includes two major components:

Mobile Server—This middle-tier infrastructure server acts as a gateway between mobile devices (such as PDAs, cellular phones, automotive computers, as well as traditional laptops) and the e-business application services those devices need to

access. Mobile Server provides the necessary functions required to support mobile, disconnected devices.

Mobile Development Kit—This SDK provides the facilities, tools, APIs and sample code to develop mission-critical mobile, disconnected applications.

Developers have the option to choose amongst three different application models:

- **Native**—Applications use ODBC to access the Oracle Lite Database on the mobile device. C++, Visual Basic for Windows CE (ADOCE), Satellite Forms or Oracle Forms are typically used to build native, device-specific applications. A native application developed in C++ can be recompiled and run on multiple devices without recoding.
- **Java**—These Java applications invoke JDBC functions to access the Oracle Lite Database on mobile devices. The UI can be built using AWT or SWING classes. Java provides reusability and cross-platform capability which makes it the choice for applications which must run on multiple devices.
- **Web**—Enables developers to run existing web applications using the J2EE Java Servlets / JSP in a disconnected mode without modifying the code base. This unique feature makes it very easy to extend web applications to mobile, disconnected devices.

Mobile Development Kit supports Windows, Palm Computing, EPOC, and different flavors of Windows CE such as Windows CE 2.11, 2.12, Pocket PC, on different chip sets such as StrongARM, MIPS, SH4, and x86.

For more information, see otn.oracle.com / products / lite.

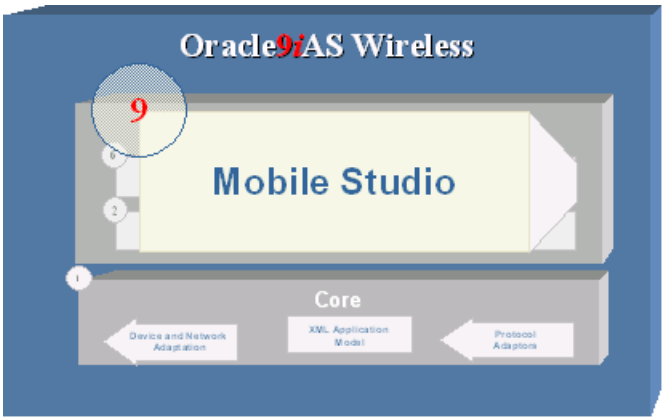
17

Mobile Studio

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

- Section 17.2, "Getting Started" introduces the Studio to new users and briefly explains how to use it to create applications.
- Section 17.3, "Studio Configuration" explains how to use the web-based administrator user interface to configure basic features of the Studio.
- Section 17.4, "Administration" explains how to use the resource-driven user interface framework to customize the look-and-feel of the Studio.
- Section 17.5, "Advanced Customization (Studio Tag Library)" explains how to use the JSP tag library to customize and enhance the layout, flow, and functionality of the Studio.
- For up-to-date information on the Oracle9*i*AS Wireless Mobile Studio, visit the OracleMobile Online Studio at http://otn.oracle.com (under Hosted Development > OracleMobile Online Studio).

Figure 17–1 Mobile Studio



17.1 Oracle9*i*AS Wireless Mobile Studio Overview

This chapter introduces Oracle9*i*AS Wireless Mobile Studio. Oracle9*i*AS Wireless Mobile Studio is a 100% online, hosted environment for developing, testing, and deploying mobile applications for the Oracle9*i*AS Wireless platform. It also serves as a web portal, supporting the wireless developer community in the enterprise and on the Internet.

The Studio offers developers a simple, intuitive, easy-to-use web-based user interface to facilitate rapid configuration, testing, and deployment of Oracle9*i*AS Wireless XML applications. Developers need not download or install anything on their workstations; all they need is a web browser and access to the Studio server. Once the XML content is registered with the Studio, the developer can test their application using any mobile device or simulator (including voice), and can instantly access debug log information in the Studio. Once the application is successfully tested, the developer may choose to deploy it to a production server with the click of a button.

Service providers can brand the Studio and customize its look-and-feel and content in order to integrate it with their existing site. The Studio's intuitive administrator interface allows web masters to rapidly create a compelling developer portal that can serve both as an interactive development tool and as a one-stop source for up-to-date information and collateral on the wireless server platform. This makes it easy for service providers to support their developer community and attract new developers.

Key features for developers are:

- 100% online, hosted environment (nothing to download).
- Simple, web-based UI targeted at application developers (as opposed to the Webtool, which is targeted at system administrators and advanced developers).
- Instant access to developed applications from any device or simulator (including voice).
- Instant debug log access for interactive testing.
- (Optional) One-click deployment to production.

Key features for service providers are:

- Serves as developer portal attracting new developers while supporting existing ones.
- Easy to brand & customize using web-based administrator UI.
- Can support multiple look-and-feel settings in multiple languages and character sets out of the box.
- Targeted at web masters, not engineers (for simple customization, no coding required; for complex customization, only HTML knowledge required).

17.2 Getting Started

17.2.1 Login and Registration

Once Oracle9*i*AS Wireless has been successfully installed, the Studio is immediately available for use without any further configuration. Use any web browser (e.g. Netscape 6.2 or Internet Explorer 6.0) to access the Studio main page at the following URL:

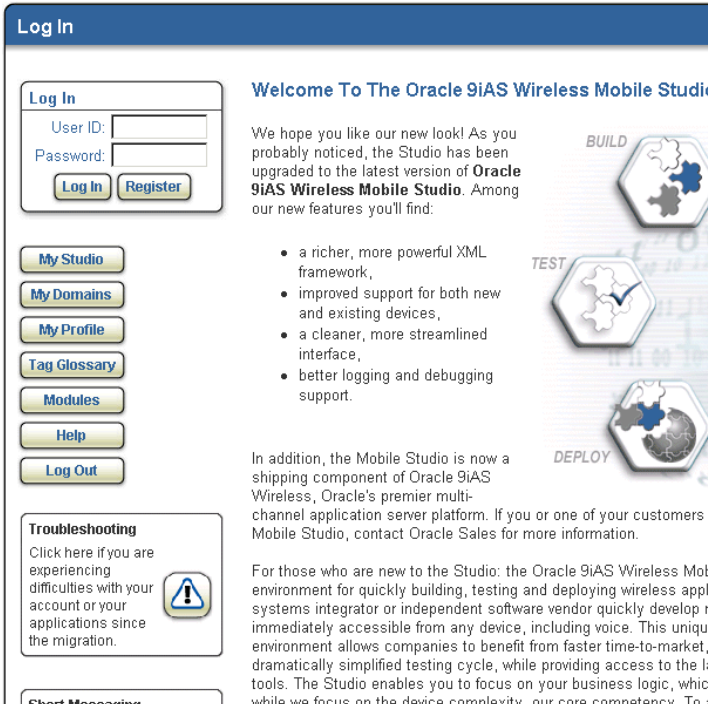
http://**<studio_server>:<studio_port>/studio**

where **<studio_server>** and **<studio_port>** are the name of the host running the Studio instance and the dedicated Studio port number, as configured in the Oracle Installer during the installation process.

Note: The Studio has been optimized for the latest versions of the popular Netscape and Internet Explorer browsers. In particular, the Studio is not certified for Netscape 4.x or Internet Explorer 4.x.

If Oracle9*i*AS Wireless was installed correctly, the above URL takes you to the Studio login screen:

Figure 17–2 Login page



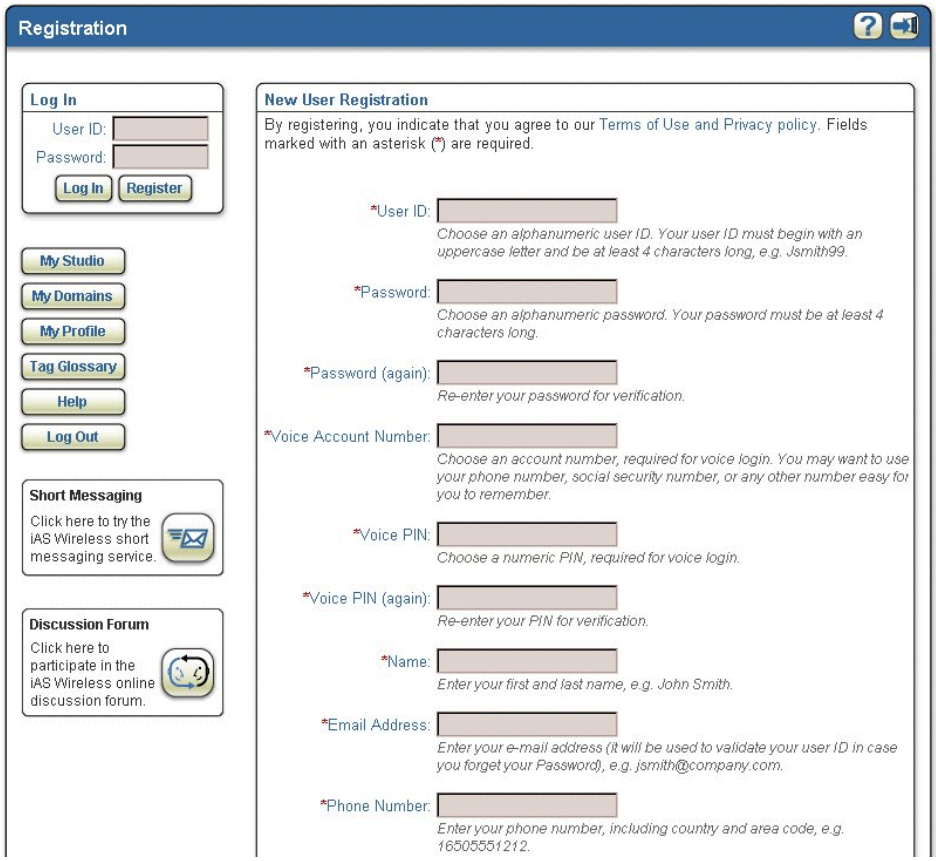
From this screen, users can:

- **Log in** to the Studio using an existing user ID and password by entering the user ID and password and clicking the Log In button.
- **Register** for a new Studio account by clicking the Register button.
- Go directly to often-used Studio pages (i.e. **My Studio**, **My Domains**, and **My Profile**) if already logged in.
- Browse the Oracle9*i*AS Wireless XML **tag glossary** by clicking the Tag Glossary button.
- Access **online help** by clicking the Help button.
- **End the session** by clicking the Log Out button.

In addition, the sample Studio includes a shortcut to the **Short Messaging** demo page (requires login) as well as to the Oracle9*i*AS Wireless **discussion forum** on the Oracle Technology Network (requires free OTN account).

For users without an existing account, clicking on the **Register** button brings up the registration page:

Figure 17–3 Registration page



Note: Single Sign-on (SSO): User profile information (including user ID and password) is stored in an Oracle Internet Directory (OID) repository and is shared by all SSO-enabled applications, including Oracle9*i*AS Wireless and all its components (such as Oracle Portal, etc.).

Additional configuration steps are required to enable single sign-on (SSO) support for the Studio. See Section 17.3, "Studio Configuration" for details.

17.3 Studio Configuration

To run the studio application successfully, you must configure certain parameters.

These parameters can be edited using the Mobile Studio Administration UI at this URL: http://myserver.myCompany.com:myport/studio/admin/config.jsp

Table 17–1 Parameters that must be configured

Name	Description	Example
deploy.ptg.url	Contains the URL of the remote Oracle9*i*AS Wireless server, to which the studio applications are deployed.	http://myserver.myCompany.com:myport/studio
samples.source.root.path	Contains the path to the folder which contains all the sample JSPs.	<WIRELESS_HOME>/wireless/j2ee/applications/studio/studio-web/samples

17.3.1 Sample Applications Configuration

New sample applications can be added to Mobile Studio using the Mobile Studio Administration UI.

Adding a sample applications involves following steps:

1. Create a sample JSP file that is Mobile XML compliant.
2. Store the sample JSP file in the samples root folder (this is same as the samples.source.root.path parameter listed above).

3. Create a new Sample Application using the Mobile Studio Administration UI.

Table 17–2 Mobile Studio JSPs

JSP	JSP	JSP	JSP
login.jsp	loginPortlet.jsp	pageMenu.jsp	pagePortlets.jsp
loginInfoBox.jsp	home.jsp	createService.jsp	editService.jsp
deployService.jsp	deployedServiceList.jsp	profile.jsp	domains.jsp
registration.jsp	newFolder.jsp	editFolder.jsp	moveOrCopy.jsp
sendMessage.jsp	viewLog.jsp	quicklink.jsp	samplesSource.jsp
pageHeader.jsp	pageFooter.jsp		

17.3.1.1 login.jsp

This is the first page users see when opening the Mobile Studio URL; users can log in Mobile Studio via this page.

If users try to access other pages without logging in, they will be redirected to this page, and instructed to log in before proceeding.

Users can take these actions from this page:

- Log in the Studio.
- Register as a new user.
- Browse through the information pages using the menu.

Figure 17–4 Login page

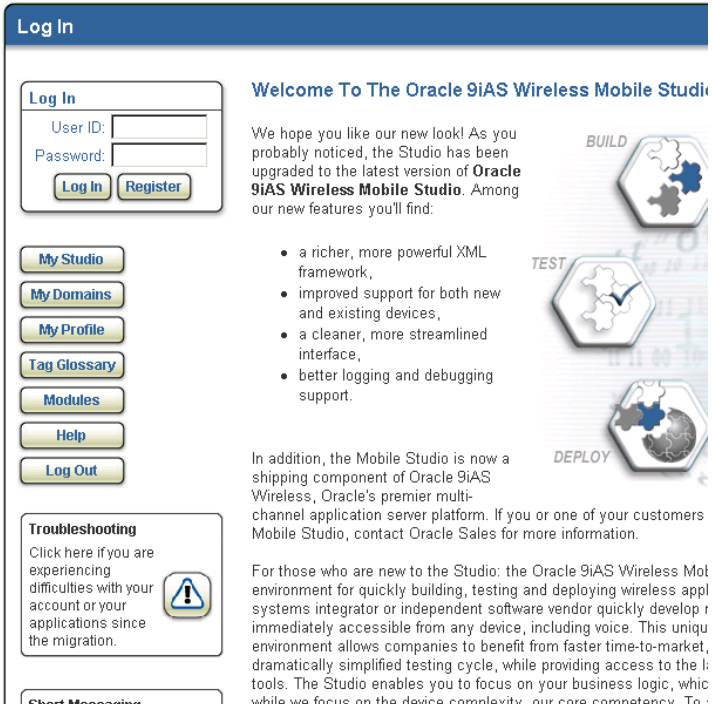


Table 17–3 Resources

Name	Description	Example
login.text.info	informational text	Oracle9*i*AS Wireless Mobile Studio is an online environment for quickly building, testing and deploying wireless applications.
login.text.title	page title	Welcome to Oracle9*i*AS Wireless Mobile Studio.
login.text.boxinfo	informational text in box	You can use Oracle9*i*AS Wireless Mobile Studio to write a single application that can be accessed via both wireless and voice interfaces.

Table 17–3 Resources

Name	Description	Example
login.text.boxtitle	box title	Developing Voice Applications
login.image.frontpage	340x340 splash image on login page	/images/frontpage.gif

Table 17–4 Action Fields

Name	Type	Description	Condition
UserId	Input field	ID for the user.	Must be a valid user ID.
Password	Input field	Password for the user.	Must be a valid password.
Login	Button	Submits the login form.	Userid and password fields must not be empty.
Register	Link	Directs user to registration page.	

17.3.1.2 loginPortlet.jsp

This page is included inside login.jsp. It contains a form for the user to complete for login.

Figure 17–5 loginPortlet

Table 17–5 Resources

Name	Description	Example	common.href.login
URL for the login page	login.jsp	common.label.login	label for login button

Table 17–5 Resources

Name	Description	Example	common.href.login
Log In	common.label.password	label for password text input field	Password
common.label.register	label for register button	Register	common.label.userid
label for userID text input field	User ID	common.title.login	portlet title text
Log In			

17.3.1.3 pageMenu.jsp

This page is included inside login.jsp. It provides a link to different pages for users to browse.

Figure 17–6 pageMenu.jsp

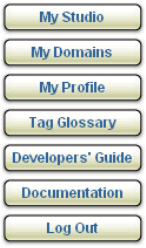


Table 17–6 Resources

Name	Description	Example	common.label.home
Label for home button.	My Studio	common.label.domains	Label for domains button.
My Domains	common.label.profile	Label for profile button.	My Profile
common.label.glossary	Label for Tag glossary button.	Tag Glossary	common.label.help
Label for documentation button.	Documentation	common.label.logout	Label for logout button.
Log Out	common.href.home	Link to home page.	home.jsp
common.href.domains	Link to domains page.	domains.jsp	common.href.profile

Table 17–6 Resources

Name	Description	Example	common.label.home
Link to profile page.	profile.jsp	common.href.glossary	Link to Tag glossary page.
tagglossary.htm	common.href.help	Link to documentation page.	openHelpwindow()
common.href.logout	Link to logout page.	logout.jsp	

Table 17–7 Action Fields

Name	Type	Description	Condition
My Studio	Link	Directs user to Home page.	User should login first.
My Domains	Link	Directs user to Domain page.	User should login first.
My Profile	Link	Directs user to Profile page	User should login first.
Tag Glossary	Link	Directs user to Tag Glossary (wealth of information on Tags)	
Help	Link	Directs user to help page.	
Log Out	Link	Logs user out.	

17.3.1.4 pagePortlets.jsp

This page is included inside login.jsp. It displays 2 portlets:

- Short Messaging
- Discussion Forum

Figure 17–7 pagePortlets page

Table 17–8 Resources

Name	Description	Example
common.title.sms	The title for short messaging	Short Messaging
common.text.sms	Informational text.	Click here to try the Oracle9*i*AS Wireless short messaging service.
sms.link.name	URL for short message page.	sendMessage.jsp
common.image.icon.sms	Image for the sms.	/images/icon_sms.gif
common.title.forum	Title of the forum portlet	Discussion forum
common.text.forum	The text for the forum portlet.	Click here to participate in the Oracle9*i*AS Wireless online discussion forum.
common.href.forum	The URL of the forum.	discuss.jsp
common.image.icon.forum	The image for the forum.	/images/icon_forum.gif

Table 17–9 Action Fields

Name	Type	Description
Condition	Short messaging	Link
Directs user to Short messaging page.	User must be logged in first.	Discussion Forum
Link	Directs user to discussion forum.	

17.3.1.5 home.jsp

This is the main user page; it is the first page that a user sees when logging in. These are the Actions available from the home.jsp page

Table 17–10 Actions available from home.jsp page.

- Create Service
- Edit Service
- Deploy Service
- Create Folder
- Rename Folder
- View Log
- View Profile
- View domains
- Explore the folders and its contents
- View sample services

Table 17–11 Resources

Name	Description	Example
home.text.greeting	Welcome message for the user.	Welcome User1
common.href.viewlog	URL of the log page	viewLog.jsp
common.href.home	URL of the home page.	home.jsp
home.image.currfold	Image for the current folder. (open folder image)	/images/folderopen.gif
home.tooltip.upfolder	The informational message to the user for the upper level folder.	Up One Level
common.href.createfolder	URL for creating new folder page.	newFolder.jsp
home.tooltip.newfolder	The informational message to the user for new folder.	Create New Folder.
home.label.newfolder	The label that appears on new Folder button.	New Folder

Table 17–11 Resources

Name	Description	Example
home.tooltip.newapp	The informational message to the user for new application.	Create New Application.
home.label.newapp	The label that appears on new Application button.	New Application
home.text.appsinfo	Informational text about applications.	Use this portlet to manage your Studio applications. To create a new application, click on the **New Application** button above.
home.label.appname	The name of the application.	Service1
home.label.appdesc	The description for the application.	This is a test service.
home.label.appstatus	The status of the application.	Deployed.
home.image.application	The icon for the Application.	/images/app.gif
home.label.view	The label on View button	View.
home.label.rename	The label on Rename button	Rename
home.label.move	The label on Move button	Move
home.label.copy	The label on copy button	Copy
home.label.delete	The label on Delete button	Delete
home.label.deploy	The label on Deploy button	Deploy
home.image.viewlog	The image for ViewLog.	images/viewlog.gif
home.text.samples	The samples application title.	Sample Applications
home.text.samplesinfo	The information about sample applications.	Use this portlet to view sample Studio applications. To view a sample application, simply select it from the list below and click on the **View** button.
home.text.test.boxtitle	The test applications title.	Testing Studio Applications.

Table 17–12 Java Beans

Name	Type	Description
foldersList	java.util.ArrayList	This list holds the list of all the sub -folders of the user for the current folder.
servicesList	java.util.ArrayList	This list holds all the services of the user for the given folder.
sampleServices	java.util.Vector	This list holds the sample services for the given user.

Table 17–13 Action Fields

Name	Type	Description	Condition
Up	Button	Moves to one level up folder.	
New Folder	Button	Directs user to create folder page.	
New Application	Button	Directs user to create Service page.	
View	Button	If Application then displays its contents. If folder then displays its sub folders and applications.	
Rename	Button	Changes the name of the folder or service.	Same name should not be used by other folder or service at same level.
Move	Button	Moves a folder / service to other folders.	Cannot move to same folder.
Copy	Button	Copies a folder / service to other folders.	Cannot copy to same folder.
Delete	Button	Deletes a given folder / service.	

Table 17–13 Action Fields

Name	Type	Description	Condition
Deploy	Button	Deploys the given Application to the Deployment server.	The URL of the deployment server should be valid.
Refresh	Button	Updates the contents.	

17.3.1.6 createService.jsp

Create a new service based on the information provided by the user.

Table 17–14 Resources

Name	Description	Example
createService.body.title	The body title for creating applications.	Create Application
common.labels.Name	The label for name field.	Name
editService.labels.name.hint	The hint for name field.	A short name for your Application.
common.labels.RemoteURL	The label for Remote URL field.	Remote URL
editService.labels.RemoteURL.hint	The hint for Remote URL field.	The URL of XML document that implements your application.
common.labels.Comments	The label for Comments.	Comments
editService.labels.comments.hint	The hint for comments field.	Comments about your application for your convenience.
common.labels.Keywords	The label for keywords.	Keywords
createService.labels.keywords.hint	The hint for keywords field.	Search keywords for your application.
common.labels.Description	The label for Description field.	Description.
common.labels.description.hint	The hint to the user for the description field	A short description for your application.
common.buttons.label.Create	The label for Create Button.	Create
common.buttons.label.Cancel	The label for Cancel Button.	Cancel

Table 17–15 Action Fields

Name	Type	Description	Condition
Name	Input	Enter the name of the application.	Cannot be left empty
URL	Input	Enter the URL of the application	Cannot be left empty
Description	Input	Enter the description for application.	
Keywords	Input	Enter the keywords.	
Comments	Input	Enter the comments.	
Create	Button	Submit the form for creating application.	
Cancel	Button	Go back to home page.	

17.3.1.7 editService.jsp

Users can view the application details and edit them.

Figure 17–8 edit application page

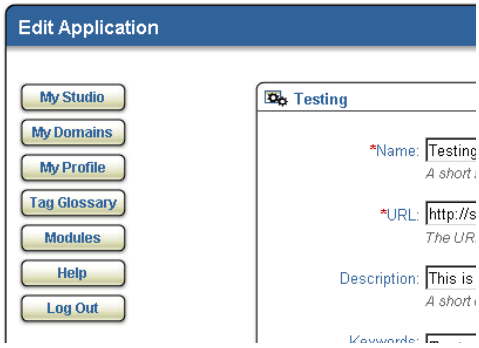


Table 17–16 Resources

Name	Description	Example
common.labels.Name	The label for name field.	Name

Table 17–16 Resources

Name	Description	Example
editService.labels.name.hint	The hint for name field.	A short name for your Application.
common.labels.RemoteURL	The label for Remote URL field.	Remote URL
editService.labels.RemoteURL.hint	The hint for Remote URL field.	The URL of XML document that implements your application.
common.labels.Description	The label for Description field.	Description.
common.labels.description.hint	The hint to the user for the description field	A short description for your application.
common.links.Comments	The label for Comments.	Comments
editService.labels.comments.hint	The hint for comments field.	Comments about your application for your convenience.
common.labels.Keywords	The label for keywords.	Keywords
editService.labels.Keywords.hint	The hint for keywords field.	Search keywords for your application.
common.buttons.label.Save	The label for Save Button.	Save
common.buttons.label.Cancel	The label for Cancel Button.	Cancel

Table 17–17 Java Objects

Name	Type	Description
coreService	oracle.panama.studio.core.CoreService	Stores all the Oracle9*i*AS Wireless service details.
studioService	oracle.panama.studio.db.StudioService	Stores studio specific details.

Table 17–18 Action Fields

Name	Type	Description	Condition
Name	Input	Edit the name of the application.	Cannot be left empty

Table 17–18 Action Fields

Name	Type	Description	Condition
URL	Input	Edit the URL of the application	Cannot be left empty
Description	Input	Edit the description for application.	
Keywords	Input	Edit the keywords.	
Comments	Input	Edit the comments.	
Save	Button	Submit the form for saving the changes.	
View Log	Button	View the runtime log for the application.	
Cancel	Button	Go back to home page.	

17.3.1.8 deployService.jsp

Enables users to deploy applications in their default domains. If an application is already deployed, then it is redeployed in that domain. In this way, if the name of an application is changed between 2 deployments, there will still be only one deployed application on the deployment server.

Figure 17–9 deploy service page

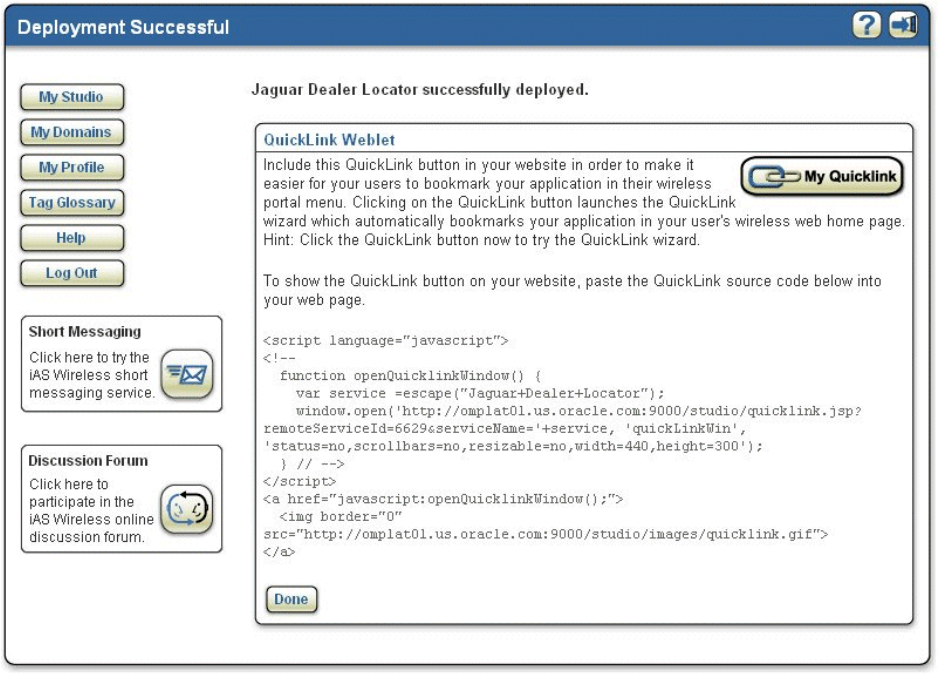


Table 17–19 Resources

Name	Description	Example
deploy.body.title	The body title text for the page.	Deploy your application.
common.labels.Application.Name	The label for application name field.	Application Name.
common.labels.Mobile.Domain	The label for domain name field.	Domain Name.
common.labels.MobileURL	The label for Mobile URL field.	Mobile URL
common.labels.Description	The label for description field.	Description
common.links.Comments	The label for comments field.	Comments

Table 17–19 Resources

Name	Description	Example
common.labels.Keywords	The label for keywords field.	Keywords
common.buttons.label.Deploy	The label for deploy button.	Deploy
common.buttons.label.Cancel	The label for cancel button.	Cancel

Table 17–20 Action Fields

Name	Type	Description	Condition
Description	Input	Edit the description for application.	
Keywords	Input	Edit the keywords.	
Comments	Input	Edit the comments.	
Comments	Button	View all the comments for the application.	
Deploy	Button	Submit the form for application deployment.	
Cancel	Button	Go back to home page.	

17.3.1.9 deployedServiceList.jsp

This page displays the list of the deployed services.

Available actions from this page:

- undeploy service
- view quicklinks
- view the list of deployed services

Figure 17–10 deployedServiceList page

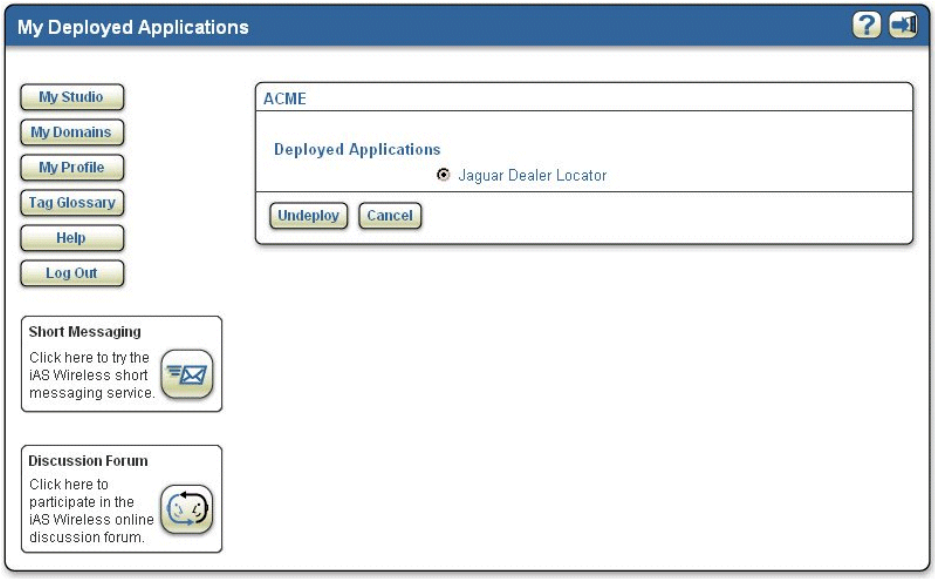


Table 17–21 Resources

Name	Description	Example
	The label for undeploy button	Undeploy
	The label for view quicklink button	View quicklink
common.buttons.label.Cancel	The label for cancel button	Cancel

Table 17–22 Action Fields

Name	Type	Description	Condition
Undeploy	Button	Submit the form for application undeployment.	Service should be selected first, and that service should be owned by the user.

Table 17–22 Action Fields

Name	Type	Description	Condition
Cancel	Button	Go back to domains page.	

17.3.1.10 profile.jsp

Users can view and edit their profiles.

Figure 17–11 profile page

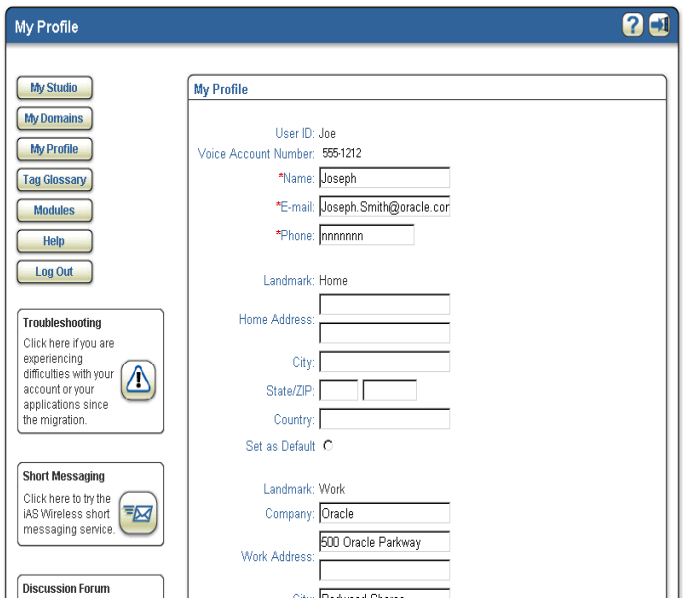


Table 17–23 Resources

Name	Description	Example
profile.body.title	The body title text for the page.	My Profile
common.labels.User.ID	The label for users id.	User ID
register.label.accountnumber	The label for account number	123456
common.labels.Name	The label for Name field	Name

Table 17–23 Resources

Name	Description	Example
common.labels.email.address	The label for email field	Email
common.labels.Phone.Number	The label for Phone field	Phone
common.labels.Landmark	The label for landmark field	Landmark
common.labels.HOME	The label for home field	HOME
common.labels.home.address	The label for home address field	Home Address
common.labels.city	The label for city field	City
common.labels.stateZip	The label for state, zip field	State / Zip
common.labels.country	The label for country field	Country
common.labels.setDefault	The label for set default field	Set as Default
common.labels.WORK	The label for work field	WORK
common.labels.Company	The label for company field	Company
common.labels.work.address	The label for work address field	Work Address
common.labels.changePin	The label for change password field	Change Password
common.labels.NewPassword	The label for new password field	New Password
common.labels.NewPasswordAgain	The label for new password again field	New Password (again)
profile.text.changepin	The label for change pin field	Change PIN
register.label.voicepin	The label for voice pin field	Voice PIN
register.label.voicepinagain	The label for voice pin again field	PIN (again)
common.buttons.label.Save	The label for save button.	Save
common.buttons.label.Cancel	The label for cancel button	Cancel

Table 17–24 Action Fields

Name	Type	Description	Condition
Name	Input	Edit the name.	
Email	Input	Edit the email.	Cannot be left empty.
Phone	Input	Edit the Phone.	Cannot be left empty.
Home Address	Inputs	Edit the home address.	
Work Address	Inputs	Edit the work address.	
Password	Input	Change the password.	Cannot be left empty.
PIN	Input	Change the PIN.	Cannot be left empty.
Save	Button	Submit the form for saving the changes.	
Cancel	Button	Go back to home page.	

17.3.1.11 domains.jsp

Allows users to create, join, delete, and manage their domains. Displays a list of domains that users own or have joined.

Figure 17–12 domains page

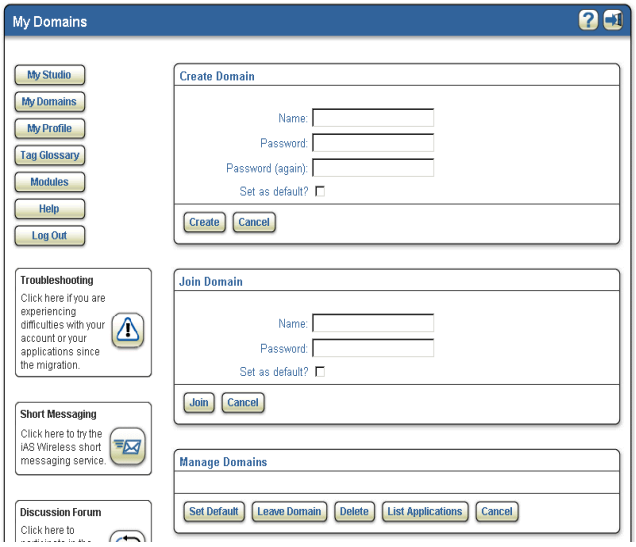


Table 17–25 Resources

Name	Description	Example
domain.body.title	The body title text for the page.	My Domain
domain.labels.new.domain	The label for new domain title	Create New domain
common.labels.Name	The label for name field	Name
common.labels.Password	The label for password field	Password
common.labels.PasswordAgain	The label for password again field	Password (again)
common.buttons.label.Create	The label for create button.	Create
common.buttons.label.Cancel	The label for cancel button	Cancel
common.labels.setDefaultQ	The label for set default field	Set as default
domain.labels.joinDomain	The label for join domain title.	Join Domain

Table 17–25 Resources

Name	Description	Example
common.buttons.label.Join	The label for name field	Name
common.labels.owneddomains	The label for owned domains field	Domains Owned.
common.labels.joineddomains	The label for joined domains field	Domains Joined.
common.buttons.label.SetDefault	The label for set default button.	Set Default
common.buttons.label.LeaveDomain	The label for leave domain button	Leave Domain
common.buttons.label.Delete	The label for delete button	Delete
common.buttons.label.LISTAPPLICATIONS	The label for list applications button	List Applications
common.buttons.label.Cancel	The label for cancel button	Cancel

Table 17–26 Action Fields: Create Domain

Name	Type	Description	Condition
Name	Input	The name of the domain to be created	Cannot be left empty
Password	Input	The password of the domain to be created	Cannot be left empty
Set as Default	Checkbox	Set the domain as default	
Create	Button	Submit the form for new domain creation.	
Cancel	Button	Go back to home page.	

Table 17–27 Action Fields: Create Domain

Name	Type	Description	Condition
Name	Input	The name of the domain to be Joined	Cannot be left empty
Password	Input	The password of the domain to be Joined	Cannot be left empty
Set as Default	Checkbox	Set the domain as default	
Create	Button	Submit the form for joining domain.	
Cancel	Button	Go back to home page.	

.

Table 17–28 Action Fields: Manage Domains

Name	Type	Description	Condition
Set Default	Button	Set the selected domain as the default one.	
Leave Domain	Button	Leave the selected domain.	The selected domain should not be the one owned.
Delete	Button	Deletes the selected domain.	The selected domain should be the owned by the user.
List Applications	Button	List the applications for the selected domain	
Cancel	Button	Go back to home page.	

17.3.1.12 registraton.jsp

Users can register to Mobile Studio by filling in the registration form.

Figure 17–13 registration page

Table 17–29 Resources

Name	Description	Example
common.href.register	The URL for registration page	Registration.jsp
register.text.title	The body title text for the page.	New User Registration
register.text.info	The informational message for registration	By registering, you indicate that you agree to our Terms of Use and Privacy policy. Fields marked with an asterisk (*) are required.
common.label.userid	The label for user id field	User ID
register.hint.userid	The hint for user id field.	Choose an alphanumeric userid.
common.label.password	The label for password field	Password
register.hint.password	The hint for password field	Choose an alphanumeric password.
common.label.password2	The label for password again field	Password (again)
register.hint.password2	The hint for password again field	Re-enter your password for verification.
register.label.accountnumber	The label for account number field	Voice account number

Table 17–29 Resources

Name	Description	Example
register.hint.accountnumber	The hint for account number	Choose an account number required for voice login.
register.label.voicepin	The label for voice pin	Voice PIN
register.hint.voicepin	The hint for voice pin	Choose a numeric PIN, required for voice login.
register.label.voicepinagain	The label for voice pin again.	PIN (again)
register.hint.voicepin2	The hint for voice again pin	Re-enter your PIN for verification.
register.hint.name	The hint for name field	Enter your first and last name, for example, John Smith.
common.label.email	The label for Email field	Email Address
register.hint.email	The hint for email	Enter your email address, for example, jsmith@company.com
common.label.phone	The label for phone field	Phone Number
register.hint.phone	The hint for phone field	Enter your phone number, including country and area code. for example, 16505151212
common.label.workaddr	The label for work address	Work Address
common.label.company	The label for company	Company Name
common.label.addr	The label for address line 1	Address Line 1
common.label.addr2	The label for address line	Address Line 2
common.label.city	The label for city.	City
common.label.state	The label for state	State
common.label.zip	The label for zip	Zip
common.label.country	The label for country	Country
register.label.setdefault	The label for set as default	Set as Default
register.hint.workaddr	The hint for work address	Enter your work address.
common.label.homeaddr	The label for home address	Home Address

Table 17–29 Resources

Name	Description	Example
register.hint.homeaddr	The hint for home address	Enter your home address.
common.label.register	The label for register button	Register
common.label.cancel	The label for cancel button	Cancel

Table 17–30 Action Fields

Name	Type	Description	Condition
User Id	Input	Enter the user id.	Cannot be left empty. Should be unique.
Account Number	Input	Enter the account number.	Cannot be left empty. Should be unique.
Name	Input	Enter the name.	
Email	Input	Enter the email.	Cannot be left empty. Should be unique.
Phone	Input	Enter the Phone.	Cannot be left empty. Should be unique.
Home Address	Inputs	Enter the home address.	
Work Address	Inputs	Enter the work address.	
Password	Input	Enter the password.	Cannot be left empty.
PIN	Input	Enter the PIN.	Cannot be left empty.
Register	Input	Submit the form for registration.	
Cancel	Input	Go back to login page.	

17.3.1.13 newFolder.jsp

Users can create new folders under their home folder or any of its sub-folders.

Figure 17–14 new folder page

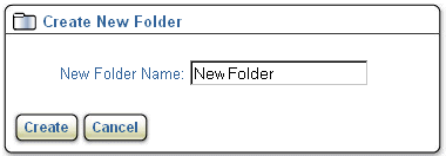


Table 17–31 Resources

Name	Description	Example
home.image.folder	Image of the folder	/images/omFolder.gif
newfolder.text.title	Title for the portlet	Create New Folder
newfolder.label.foldername	The label for the folder name	New Folder Name
newfolder.text.foldername	The text for the folder	New Folder
newfolder.label.create	The label for create button	Create
newfolder.label.cancel	The label for cancel button	Cancel

Table 17–32 Action Fields

Name	Type	Description
Condition	Folder Name	Input
The name of the new folder.	Cannot be left empty and should be unique	Create
Button	Submit the form for creating folder.	
Cancel	Button	Go back to home page.

17.3.1.14 editFolder.jsp

Allows users to rename their folders or applications.

Figure 17–15 edit folder page

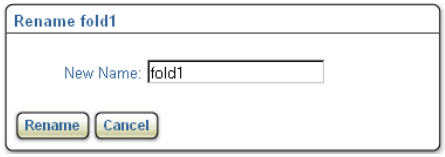


Table 17–33 Resources

Name	Description	Example
rename.text.title	Title for the portlet	Rename fold1
rename.label.newname	Label for the name of folder	New Name
rename.label.rename	The label for the rename button	Rename
rename.label.cancel	The label for the cancel button	Cancel

Table 17–34 Action Fields

Name	Type	Description	Condition
Name	Input	The name of the folder or service.	Cannot be left empty and should be unique
Rename	Button	Submit the form for renaming folder or service.	
Cancel	Button	Go back to home page.	

17.3.1.15 moveOrCopy.jsp

Allows users to move /copy their folders or services.

Figure 17–16 move/copy pages

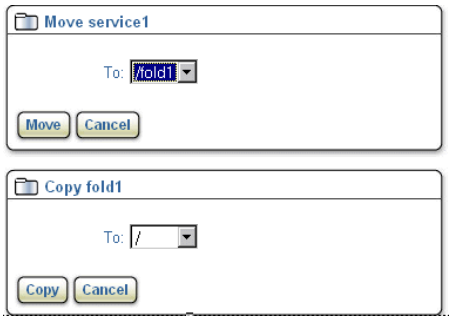


Table 17–35 Resources

Name	Description	Example
home.image.folder	The image of the folder	/images/omFolder.gif
move.label.to	The label for the destination folder	To
move.label.cancel	The label for the cancel button	Cancel

Table 17–36 Action Fields

Name	Type	Description	Condition
To	Select	Users folders list.	Cannot be left empty and should be unique
Move	Button	Submit the form for moving folder/service.	Cannot move to same folder.
Copy	Button	Submit the form for copying folder/service.	Cannot copy to same folder.
Cancel	Button	Go back to home page.	

17.3.1.16 sendMessage.jsp

Users can send messages to a number of people using Email, SMS, Voice, Two way pager and other supporting features that are available in Oracle9*i*AS Wireless.

Figure 17–17 send message page

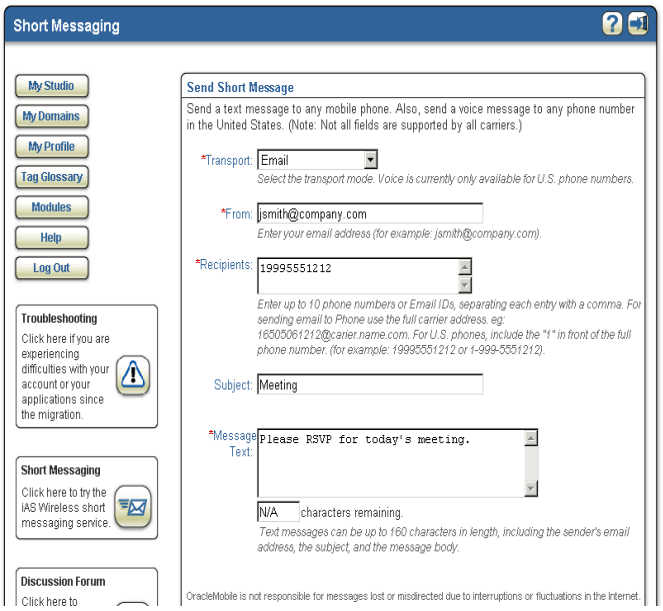


Table 17–37 Resources

Name	Description	Example
common.href.sms	The url for the send massage page	sendMessage.jsp
sms.text.title	The title for the page	Short Messaging
sms.text.info	The informational text.	Send a text message to any mobile phone. Also, send a voice message to any phone number in the United States.
sms.label.transport	The label for transport field	Transport

Table 17–37 Resources

Name	Description	Example
sms.hint.transport	The hint for transport field	Select the transport mode. Voice is currently only available for US phone numbers.
sms.label.sender	The label for sender field	From
sms.hint.sender	The hint for sender field	Enter your email address.
sms.label.recipients	The label for recipients field	Recipients
sms.hint.recipient	The hint for recipients field.	Enter up to 10 phone numbers, separating each entry with a comma.
sms.label.subject	The label for subject field	Subject
sms.label.message	The label for message field	Message Text
sms.hint.messagechar	The text for characters remaining	Characters remaining
sms.hint.message	The hint for message field	Text messages can be upto160 characters in length, including the senders email address, the subject and the message body.
sms.text.legal	The text for legal terms	Oracle mobile is not responsible for messages lost or misdirected due to interruptions or fluctuations in internet.
sms.label.sendmessage	The label for send message button	Send Message

Table 17–38 Action Fields

Name	Type	Description	Condition
Transport	Select	List of the supported transport type.	
From	Input	Senders email Id / Phone number.	

Table 17–38 Action Fields

Name	Type	Description	Condition
Recipients	Input	Submit the form for copying folder / service.	Cannot be left empty
Subject	Input	Subject of the message.	
Message Text	Input	Message text.	
Send Message	Button	Submit the form for sending message.	

17.3.1.17 viewLog.jsp

Users can test their services by viewing the log message for the service at runtime.

Figure 17–18 view log page

Table 17–39 Resources

Name	Description	Example
viewlog.head.title	The head title for view log page.	View Log
viewlog.body.title	The body title for view log page.	View Log Messages
home.image.viewlog	The image for view log	images / viewlog.gif

Table 17–39 Resources

Name	Description	Example
viewlog.label.close	The label for close button	Close
viewlog.label.show	The text	Show
viewlog.label.requests	The text message	Most recent log messages
viewlog.label.refresh	The label for refresh button	Refresh Log

Table 17–40 Action Fields

Name	Type	Description	Condition
No. of Logs	Input	Enter the no. of log messages to see.	
Refresh Log	Input	Submits the form for refreshing logs.	
Close Window	Input	Closes the window.	

17.3.1.18 quicklink.jsp

Users can *quicklink* an application that is deployed. The next time when they log in to Oracle9*i*AS Wireless using their mobile devices, they can access this application.

Figure 17–19 quicklink page

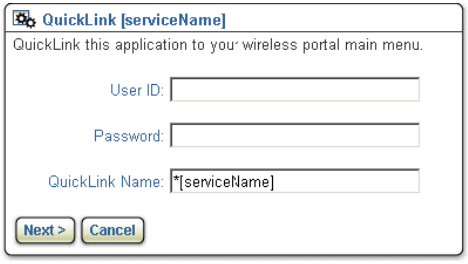


Table 17–41 Resources

Name	Description	Example
quicklink.text.title	The head title for the portlet	Quicklink

Table 17–41 Resources

Name	Description	Example
quicklink.body.title	The body title for the portlet	Processing Auto Quicklink
quicklink.text.info	Informational text	QuickLink this application to your wireless portal main menu.
quicklink.label.userid	The label for userid field	User ID
quicklink.label.password	The label for password field	Password
quicklink.label.linktitle	The label for quicklink name field	Quicklink Title
quicklink.label.cancel	The label for cancel button	Cancel
quicklink.label.next	The label for next button	Next

17.3.1.19 sampleSource.jsp

Displays the JSP source code of the sample application for users reference.

Figure 17–20 sample source page

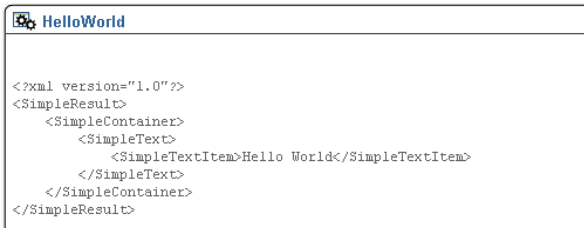


Table 17–42 Resources

Name	Description	Example
home.image.application	The image for the application	/images/app.gif
samplesrc.label.close	The label for close button	Close Window

17.3.1.20 pageHeader.jsp

This page is included as a header for all the customer facing JSPs.

Figure 17–21 page header

Table 17–43 Resources

Name	Description	Example
page.title	window title	Home
site.name	site name	
common.image.button.help	Help button image name	/images/button_help.gif
common.image.button.logout	Logout button image name	/images/button_logout.gif
common.image.window.left	left window border image name	/images/window_left.gif
common.image.window.top	top window border image name	/images/window_top.gif
common.image.window.topleft	top left window corner image name	/images/window_topleft.gif
common.image.window.topright	top right window corner image name	/images/window_topright.gif
common.href.help	URL of the Help page	documentation.jsp
common.href.logout	URL of the Logout action	logout.jsp
common.tooltip.help	tooltip text for the Help button	Help
common.tooltip.logout	tooltip text for the Logout button	Log out

17.3.1.21 pageFooter.jsp

This page is included as a footer for all customer facing JSPs.

Figure 17–22 page footer.

Table 17–44 Resources

Name	Description	Example
common.image.window.bot	bottom window border image name	/images/window_bot.gif
common.image.window.botleft	bottom left window corner image name	/images/window_botleft.gif
common.image.window.botright	bottom right window corner image name	/images/window_botright.gif
common.image.window.left	left window border image name	/images/window_left.gif
common.image.window.line	window separator line image name	/images/window_line.gif
common.image.window.lineleft	window separator left joint image name	/images/window_lineleft.gif
common.image.window.lineright	window separator right joint image name	/images/window_lineright.gif
common.image.window.right	right window border image name	/images/window_right.gif

Table 17–45 Customizable User Messages

Name	Value
error.accountnumber.exist	Account number already used. Please try another value.
error.create.user.exist	User of this name already Exist
error.create.user.generic	Unable to register user.
error.update.user.generic	Unable to update user profile.
error.create.email.exist	Email address already used by other user.
error.create.voice.exist	Phone number already used by other user.
error.login.user.param	Please log in using a valid User ID and password.
error.restricted.page.access	First login using valid UserId and Password.
error.select.service.edit	Select the service you want to edit.

Table 17–45 Customizable User Messages

Name	Value
error.select.service.deploy	Select the service you want to deploy.
error.select.service.delete	Select the service you want to delete.
error.delete.service.generic	Unable to delete the service. Please try again later.
error.create.service.generic	Unable to create the service. Please try again later.
error.create.service.exist	Service of same name already exist.
error.edit.service.generic	Unable to edit the service. Please try again later.
error.servicename.mobileurl.null	Service Name and the Mobile URL cannot be null.
error.deploy.service.generic	Unable to deploy the service. Please try again later.
error.deploy.service.name.exist	Service of same name already deployed by other user in same domain. Try other name or other domain.
error.default.domain.notexist	Currently you are not associated with any domain, either join or create one.
error.invalid.input.field	Please enter valid information in the given fields.
error.create.folder.generic	Unable to create the folder. Please try again later.
error.rename.folder.generic	Unable to rename the folder. Please try again later.
error.create.folder.exist	Folder of this name already exist.
error.copy.folder.same	Not allowed. The destination folder is same as folder.
error.move.folder.same	Not allowed. The destination folder is same as folder.
error.select.folder.rename	Select the folder you want to rename.
error.folder.name.null	Name of the Folder cannot be null.
error.select.folder.delete	Select the folder you want to delete.
error.delete.folder.generic	Unable to delete the folder. Please try again later.
error.quicklink.user.invalid	Please enter valid username and password.
error.create.quicklink.generic	Unable to quicklink the service. Please try again later.
error.get.domains.generic	Unable to fetch the domains. Please try again later.
error.domain.name.null	Domain name and password cannot be null.
error.create.domain.exist	Domain of same name already exist. try different name.
error.create.domain.generic	Unable to create domain. Please try again later.

Table 17–45 Customizable User Messages

Name	Value
error.select.domain.delete	Select the domain you want to delete.
error.delete.domain.notowner	Only the owner of the domain can delete it.
error.delete.domain.generic	Unable to delete domain. Please try again later.
error.join.domain.notexist	Invalid domain name / password.
error.join.domain.generic	Unable to join domain. Please try again later.
error.select.domain.leave	Select the domain you want to leave.
error.leave.domain.generic	Unable to leave domain. Please try again later.
error.select.domain.default	Select the domain you want to set as default.
error.deployedservice.list	No Services are deployed in this domain.
error.undeployservice.noprivilige	Only the owner of the service can undeploy it.
message.domain.not.found	No Domains found.
error.domain.alreadyowned	Domain already owned, no need to join it.
error.leave.domain.owned	You cannot leave the owned domains. Delete if not required.
error.resource.notFound	Resource "{0}" not found.
error.session.expired	Your session has expired. Please log in using a valid user ID and password.
error.login.invalidPassword	Incorrect password. Please log in using a valid user ID and password.
error.login.secureDocument	You have tried access a secure document. Please log in first.
error.user.notLoggedIn	User not logged in
error.privileges.generic	You do not have appropriate privileges.
error.create.generic	Unable to create. Please try again later.
error.rename.generic	Unable to rename. Please try again later.
error.move.generic	Unable to Move. Please try again later.
error.copy.generic	Unable to Copy. Please try again later.
error.create.folderOrService.exist	Name already used. Try different name.
error.select.generic	Please make a selection first.

Table 17–45 Customizable User Messages

Name	Value
success.edit.service	Changes saved.
success.create.service	Application created.
success.create.folder	Folder created.
success.create.domain	Domain created.
success.join.domain	Joined the requested Domain.
success.leave.domain	Left the requested Domain.
success.delete.domain	Domain deleted.
success.default.set.domain	Set Domain as default.
success.rename.service	Successfully renamed.

17.4 Administration

17.4.1 Login

Location: http://hostname:port/studio/admin/login.jsp

1. Type in user name (for example, orcladmin)
2. Type in password (for example, manager)
3. Click on 'Login'; if your user name and password are correct, you will be brought to the administration pages.

> **Note:** If you tried to access the administration pages without logging in first, you will be brought to this page automatically and the following error message will be shown on the top of the page: *"Your session has expired. Please log in using a valid user ID and password"*.

Subsequent pages can be accessed only after you have logged in.

For any of the changes that an administrator makes through the Administration UIs to be visible to end-users, the administrator must press the "*Reset*" button that is located on the top right-hand side of the pages.

17.4.2 Site

Location: http://hostname:port/studio/admin/site.jsp

From the Sites page, you can find site(s) using a pattern, or you can add, edit, and delete sites.

To find a site (if, for example, you have many sites), type the name or a pattern for the site you are looking for, then click the "Find" button. You will be given a list of the sites that matched the name or pattern.

To add a site, click the "Add" button; you will be brought to another page, http://hostname:port/studio/admin/editSite.jsp, where you can create a site after you specify the name and description and pick a default locale for the site. If you click "Save", then the changes are stored and you are brought back to the Sites page. If you click "Cancel", no changes will be stored and you will be brought back to the Sites page.

To edit a site, first select the site you want to edit from the list of sites you have, then click the "Edit" button below the list of sites. You will be brought to the another page, http://hostname:port/studio/admin/editSite.jsp, where you can make changes to the site and store the changes. For a site created by an administrator, the name, description, and default locale can all be changed. However, only description is changeable for the default site.

To delete a site, select the site you want to delete and the click the "Delete" button. The save the deletion, click the "Save" button. To undo the deletion, click the "Undelete" button. The "Undelete" button appears if you have just deleted site(s), and you have not saved your changes yet.

Note: The changes will not be stored until you click the "Save" button.

17.4.3 Configuration

Location: http://hostname:port/studio/admin/config.jsp

From the Configuration page, you can find configuration parameter(s) using a pattern, or you can add, edit, and delete configuration parameters.

To find a configuration parameter (if, for example you have many configuration parameters), type the name or a pattern for the parameter you are looking for in the textbox, and click the "Find" button. You will be given a list of the configuration parameters that match the name or pattern.

To add a configuration parameter, click the “Add” button, you will be brought to another page, http://hostname:port/studio/admin/editConfig.jsp, where you can create a configuration parameter. You must specify the name and description for a parameter before it can be created. You can add value(s) to the parameter by clicking the “Add” button. Once you have added a value to a list of values for the parameter, you can delete, or move them up/down the list.

Note: The first value on the list is used as the value of the parameter while the other values are just for convenience.

When you click the “Save” button, the changes are stored and you are brought back to the configuration parameters page. If you click the “Cancel” button, no changes will be stored and you will be brought back to the configuration parameters page.

To edit a configuration parameter, select the configuration parameter you want to edit from the list of configuration parameters, then click the “Edit” button below the list of configuration parameters. You will be brought to another page, http://hostname:port/studio/admin/editConfig.jsp, where you can make changes to the configuration parameters and store them.

To delete a configuration parameter, select the configuration parameter you want to delete and the click the “Delete” button. To save the deletion, click the “Save” button. To undo the deletion, click the “Undelete” button. The “Undelete” button appears if you have just deleted configuration parameter(s), and you have not yet saved your changes.

Note: The changes will not be stored until you click the “Save” button.

IMPORTANT:

The site you designate as the default site must be given this name:

`oracle.panama.studio.resource.defaultSite`

For a list of other import configuration parameters, see Section 17.3, "Studio Configuration" on studio configuration.

17.4.4 Locales

Location: http://hostname:port/studio/admin/locale.jsp

From the Locales page, you can either find locale(s) using a pattern, or you can add, edit, and delete locales.

To find a locale, type the name or a pattern for the locale you are looking for in the textbox, and click the “Find” button. You will be given a list of the locales that match the name or pattern.

To add a locale, click the “Add” button, a new entry will be added to the list of locales with the name and description fields left blank. You must specify the name and description for a locale before it can be created. Click the “Save” button to store the change(s).

To edit a locale, edit the name or description of the locale in the appropriate fields. Click the “Save” button to store the change(s). Note that the name of the default locale is not editable.

To delete a locale, select the locale you want to delete, then click the “Delete” button. To save the deletion, click the “Save” button. To undo the deletion, click the “Undelete” button. The “Undelete” button appears if you have just deleted configuration parameter(s), but have not yet saved your changes. Note that the default locale cannot be deleted.

Note: The changes will not be stored until you click the “Save” button.

Mobile Studio comes with default bundles for 28 different locales as listed in the following table:

Locales of the default bundles for Mobile Studio.

Table 17–46 Mobile Studio locales

Name	Description	Name	Description
ar	Arabic	ko	Korean
cs	Czech	nl	Dutch
da	Danish	no	Norwegian
de	German	pl	Polish
el	Greek	pt	Portuguese
es	Spanish	Pt_BR	Portuguese (Brazil)
es_ES	Spanish	ro	Romanian

Table 17–46 Mobile Studio locales

Name	Description	Name	Description
fi	Finnish	ru	Russian
fr	French	sk	Slovak
fr_CA	French (Canada)	sv	Swedish
hu	Hungarian	th	Thai
it	Italian	tr	Turkish
iw	Hebrew	Zh_CN	Chinese (PRC)
ja	Japanese	Zh_TW	Chinese (Taiwan)

The locales are only *enabled* after the administrator has explicitly added them through this interface and reset the system by clicking the "*Reset*" button. For example, to support users whose preferred locale is 'ru', the administrator must add 'ru' and the description (for example: "Russian"), through the UI, save the changes, and click the "*Reset*" button to see the effects.

The following is a description of the algorithm that Mobile Studio uses to resolve the locale to use:

1. Given the list of preferred locales for the user, which can be obtained from the request.
2. Iterate through the list, and for each preferred locale *L*, check if it can be found in the list of enabled locales for Mobile Studio. If *L* can be found, return the found locale and the iteration stops. If *L* cannot be found, then another search on a new *L'* which is constructed using only the language part of *L* is tried. For example, if "*en_US*" cannot be found, then "*en*" will be tried instead. If the second try works, the new locale *L'* will be returned and the iteration stops.
3. If after the iteration finished and no suitable locale can be found, then the default locale of the default site will be returned if it is enabled. Similarly, a second try will be given to this default locale as well.
4. If after step 3, the preferred locale is still not found, locale "*en*" will be returned.

If an additional locale other than these 28 locales is needed (for example, "*hi*" for Hindi), it is the administrator's responsibility to provide the following resources to make sure Mobile Studio works consistently.

- The administrator must provide a "*DefaultSite_hi.properties*" file or go through the resource administration pages to provide a value of locale "*hi*" for each of

the resources that must be changed. To add the file to the application, follow these steps:

1. From the Oracle9*i*AS Wireless root directory, navigate to iaswv20/wireless/lib, and find studio.jar file.
2. Unjar the file and add "*DefaultSite_hi.properties*" to the extracted files.
3. Jar all the files back into studio.jar.

- The administrator must provide the "*messages_hi.properties*" file for messages. To add the file to the application, follow these steps:
  1. From the Oracle9*i*AS Wireless root directory, navigate to iaswv20/wireless/server/classes/messages/oracle/panama/studio.
  2. Insert "*messages_hi.properties*" there.
- The administrator must provide the "*ommsg_hi.js*" file for javascript messages. To add the file to the application, follow these steps:
  1. From the Oracle9*i*AS Wireless root directory, navigate to iaswv20/wireless/j2ee/applications/studio/studio-web/javascript/.
  2. Insert "*ommsg_hi.js*" there.
- The instance must be restarted after these changes.

17.4.5 Sample Services

Location: http://hostname:port/studio/admin/samples.jsp

From the Sample Services page, you can either find the list of sample service(s) or you can add, edit, and delete locales.

To add a sample service, click the "Add" button; you will be brought to another page, http://hostname:port/studio/admin/editSamples.jsp. You must specify the name, description, the name of the service JSP and service URL before the new sample service can be stored. The JSP name of the service is the name of the JSP that supports the service, while the service URL is the URL used by Oracle9*i*AS Wireless server at runtime for the application. The sample service can be hidden from users by setting the Visible to "No", or it can be shown to users by setting it to "Yes". To save the changes, click the "Save" button.

To edit a sample service, click the "Edit" button; you will be brought to the http://hostname:port/studio/admin/editSamples.jsp page, where you can change the name, description, JSP name, the service URL, and the status of visibility of the sample service. Again, click the "Save" button if you want the change(s) stored.

To delete a sample service, first select the sample service from the list of services shown then click the “Delete” button.

17.4.6 Resources

Location: http://hostname:port/studio/admin/resource.jsp

From the Resources page, you can either browse the list of resources(s) or you can add, edit, and delete resources.

The resources are displayed in the fashion similar to a file directory structure. On first access to the resource page, the resources and folders inside the root folder are shown. To drill down a folder, click on the name of the folder. To go back up a level, click on the link called “Up One Level” on the front of the resource list.

To add a resource, click on the “Add” button, you will be brought to another page, http://hostname:port/studio/admin/editResource.jsp. You must specify the name, description, the type of the resource and default locale for the resource before the new resource can be created. To add a value for the resource, click the “Add” button on this page, select the site you want the resource to be bound to, the locale for the resource, and type in the value for the resource in the text box. Click the “Save” button to store the change(s). Only resources can be added to the system, however, folders can be added as a side effect. For example, if you are browsing the directory, “deploy.head”, and then you decide to add a resource called “deploy.head.folder.resource”, if the “deploy.head.folder” does not exist, it will be created in addition to the resource “deploy.head.folder.resource”.

To edit a resource, select the resource from the resource lists then click the “Edit” button. You will be brought to http://hostname:port/studio/admin/editResource.jsp, where you can change the name, description, type of the resource, and default locale for the resource.

Note: You CANNOT edit a folder at this location. For default resources, name and default value CANNOT be edited. Click the “Save” button to store the change(s).

To delete a resource, select the resource from the resources list, then click the “Delete” button.

Note: folders and default resources CANNOT be deleted. To undo the deletion, click the “Undelete” button. Click the “Save” button to store the change(s). Remember, the changes WILL NOT be stored until you click the "Save" button.

17.5 Advanced Customization (Studio Tag Library)

17.5.1 Resources

Resources currently supported in Mobile Studio are divided into two categories: *literal resources* and *http resources*.

Literal resources are resources with a literal value, as the name suggests. Mobile Studio includes a set of literal resources; advanced users of Mobile Studio can customize them, although it is an involved process and therefore recommended only for advanced users. Literal resources can always be overridden (effectively, customized) from the Mobile Studio administration pages.

Http resources are usually static HTML pages that can be dynamically included at runtime and they should be stored inside a directory that can be seen by the web server.

All the resources can be accessed using the `<om:res>` tag; the only requirement for this tag to work is that the attribute with the given name must exist in the current context (that is, the current http request, the page context, or the current session).

At runtime, the tag library automatically determines the type of the resource and outputs its value dynamically (that is, the literal value or the HTML page).

17.5.2 Tag Library

<om:is />

Specification in taglib:

```
<tag>
  <name>is</name>
  <attribute>
    <name>attr</name>
```

```
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>value</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>name</name>
      <required>false</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>prefix</name>
      <required>false</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
    <tagclass>oracle.panama.studio.taglib.IsTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Evaluate the body if the value found from the bean matches the given
value</info>
 </tag>
```

Usage:

```
<om:is attr= "attrName" value= "attrValue" (name=  "beanName") (prefix=
"prefixName") >…</om:is>
```

> **Note:** The attributes in parenthesis are optional.

`<om:is>` is a body tag, that is, if the condition evaluate is set to true, then the body content will be processed and output as appropriate. The attributes are explained as follows:

- *attr*: The attribute of the bean whose value is being tested.
- *value*: The value of the bean attribute being testing against.
- *name* (optional): The name of the bean in context.
- *prefix* (optional): When asking the bean for the attribute, the prefix to the name of the attribute is added (for example, if the prefix is "is", and the attribute is "empty", then the method to invoke on the bean would be "isEmpty()" which follows Java naming conventions). If not given, then we test first "get" is tested

first, then “is” is tested as the prefix. If neither one is found, then an exception will be thrown.

<om:not>

Specification in taglib:

```
 <tag>
   <name>not</name>
   <attribute>
     <name>attr</name>
     <required>true</required>
     <rtexprvalue>true</rtexprvalue>
   </attribute>
   <attribute>
     <name>value</name>
     <required>true</required>
     <rtexprvalue>true</rtexprvalue>
   </attribute>
   <attribute>
     <name>name</name>
     <required>false</required>
     <rtexprvalue>true</rtexprvalue>
   </attribute>
   <attribute>
     <name>prefix</name>
     <required>false</required>
     <rtexprvalue>true</rtexprvalue>
   </attribute>
   <tagclass>oracle.panama.studio.taglib.NotTag</tagclass>
   <bodycontent>JSP</bodycontent>
   <info>Evaluate the body if the value found from the bean does not match the
given value.</info>
 </tag>
```

Usage:

```
<om:not attr= “attrName” value= “attrValue” (name=  “beanName”) (prefix=
“prefixName”) >….</om:not>
```

<om:not> is a body tag (that is, if the condition evaluate is set to *false*, then the body content will be processed and output as appropriate.

The attributes are explained as follows:

- *attr*: The attribute of the bean whose value is being tested.

- *value*: The value of the bean attribute being testing against.
- *name* (optional): The name of the bean in context.
- *prefix* (optional): When asking the bean for the attribute, the prefix to the name of the attribute is added (for example, if the prefix is "is", and the attribute is "empty", then the method to invoke on the bean would be "isEmpty()" which follows Java naming conventions). If not given, then we test first "get" is tested first, then "is" is tested as the prefix. If neither one is found, then an exception will be thrown.

<om:get />

Specification in taglib:

```
<tag>
  <name>get</name>
  <attribute>
    <name>attr</name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
  <attribute>
     <name>name</name>
     <required>false</required>
     <rtexprvalue>true</rtexprvalue>
  </attribute>
  <tagclass>oracle.panama.studio.taglib.GetTag</tagclass>
  <bodycontent>empty</bodycontent>
  <info>Gets the value of a bean attribute using reflection.</info>
</tag>
```

Usage:

```
<om:get attr= "attrName" (name=  "beanName")/>
```

<om:get> is a simple tag, that is, it does not allow content inside its body. This tag tried to get the attribute from bean and output it if found. It does nothing if the bean or the attribute are not found.

The attributes are explained as follows:

- *attr*: The attribute of the bean whose value is being tested.
- *value*: The value of the bean attribute being testing against.

<om:bean />

Specification in taglib:

```
<tag>
  <name>bean</name>
  <attribute>
     <name>name</name>
     <required>true</required>
     <rtexprvalue>true</rtexprvalue>
  </attribute>
  <tagclass>oracle.panama.studio.taglib.BeanTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>Sets up the context for the bean.</info>
</tag>
```

Usage:

```
<om:bean name=  "beanName"/>
```

<om:bean> is a simple tag, that is, it does not allow content inside its body. This tag put the bean in the context (same as JSP page context) with the given name.

The attributes are explained as follows:

- *attr*: The attribute of the bean whose value is being tested.
- *value*: The value of the bean attribute being testing against.

<om:test />

Specification in taglib:

```
<tag>
  <name>test</name>
  <attribute>
    <name>attr</name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
  <attribute>
     <name>value</name>
     <required>true</required>
     <rtexprvalue>true</rtexprvalue>
  </attribute>
  <attribute>
     <name>prefix</name>
     <required>false</required>
     <rtexprvalue>true</rtexprvalue>
  </attribute>
  <tagclass>oracle.panama.studio.taglib.TestTag</tagclass>
  <bodycontent>JSP</bodycontent>
```

```
    <info>Test if the value of a bean attribute is the same as given using
reflection.</info>
  </tag>
```

Usage:

```
<om:test attr= "attrName" value= "attrValue" (prefix = "prefix")>….</om:test>
```

<om:test> is a body tag, that is, its body content is evaluated and output as appropriate if the test condition evaluates to true.

The attributes are explained as follows:

- *attr*: The attribute of the bean whose value is being tested.
- *value*: The value of the bean attribute being testing against.
- *prefix* (optional): The prefix that can be used when invoking the method against the bean in context.

<om:equals />

Specification in taglib:

```
  <tag>
    <name>equals</name>
    <attribute>
      <name>attr</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
       <name>name</name>
       <required>true</required>
       <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
       <name>prefix</name>
       <required>false</required>
       <rtexprvalue>true</rtexprvalue>
    </attribute>
    <tagclass>oracle.panama.studio.taglib.EqualsTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Test if the value of a bean attribute is the same as given using
reflection.</info>
  </tag>
```

Usage:

```
<om:equals  attr= “attrName” name= “attrName” (prefix = “prefix”)>….</om:equals>
```

<om:equals> is a body tag, that is, its body content is evaluated and output as appropriate if the test condition evaluates to true.

The attributes are explained as follows:

- *attr*: The attribute of the bean whose value is being tested.
- *value*: The value of the bean attribute being testing against.
- *prefix* (optional): The prefix that can be used when invoking the method against the bean in context.

<om:indexIs />

Specification in taglib:

```
  <tag>
     <name>indexIs</name>
     <tagclass>oracle.panama.studio.taglib.IndexIsTag</tagclass>
     <bodycontent>JSP</bodycontent>
     <info>Test if the index inside iteration is the same as given by the
user.</info>
     <attribute>
        <name>value</name>
<required>true</required>
<rtexprvalue>true</rtexprvalue>
     </attribute>
  </tag>
```

Usage:

```
<om:indexIs  value= “value” >….</om:indexIs>
```

<om:indexIs> is a body tag, that is, its body content is evaluated and output as appropriate if the test condition evaluates to true.

The attributes are explained as follows:

- *value*: The value of the bean attribute being testing against.

<om:indexEquals />

Specification in taglib:

```
  <tag>
     <name>indexEquals</name>
     <tagclass>oracle.panama.studio.taglib.IndexEqualsTag</tagclass>
     <bodycontent>JSP</bodycontent>
```

```
     <info>Test if the index inside iteration is the same as the value
found.</info>
     <attribute>
        <name>name</name>
<required>true</required>
<rtexprvalue>true</rtexprvalue>
     </attribute>
  </tag>
```

Usage:

```
<om:indexEquals  name= "attrName" >….</om:indexEquals>
```

<om:indexEquals> is a body tag, that is, its body content is evaluated and output as appropriate if the test condition evaluates to true.

The attributes are explained as follows:

- *name*: The name of the attribute in context whose value is being tested.

<om:index />

Specification in taglib:

```
  <tag>
    <name>index</name>
    <tagclass>oracle.panama.studio.taglib.IndexTag</tagclass>
    <bodycontent>empty</bodycontent>
    <info>Gets the current index during iteration.</info>
  </tag>
```

Usage:

```
<om:index/>
```

<om:index> is a simple tag, that is, there is no content inside the tag allowed. It simply outputs the index of the current bean which is used mostly inside <om:iterate>.

<om:res />

Specification in taglib:

```
  <tag>
    <name>res</name>
    <attribute>
      <name>name</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
```

```
    </attribute>
    <tagclass>oracle.panama.studio.taglib.ResourceTag</tagclass>
    <bodycontent>empty</bodycontent>
    <info>A generic "get-resource" tag.</info>
  </tag>
```

Usage:

```
<om:res name= “resName” >….</om:res>
```

<om:res> is a simple tag, that is, there is no content allowed inside its body.

The attributes are explained as follows:

- *name*: The name of the attribute in context whose value is to be output if it exists.

The context is the defined as the HTTP request parameter, the JSP page context, or the current HTTP session.

<om:enc />

Specification in taglib:

```
  <tag>
    <name>enc</name>
    <attribute>
      <name>name</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
    <tagclass>oracle.panama.studio.taglib.EncodeTag</tagclass>
    <bodycontent>empty</bodycontent>
    <info>A generic "encode-resource" tag.</info>
  </tag>
```

<om:exist />

Specification in taglib:

```
 <tag>
    <name>exist</name>
    <tagclass>oracle.panama.studio.taglib.ExistTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>A logical exists tag.</info>
    <attribute>
      <name>name</name>
      <required>true</required>
```

```
      <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>
```

Usage:

```
<om:exist name= "attrName" >….</om:exist>
```

<om:exist> is a body tag, that is, its body content is evaluated and output as appropriate if the test condition evaluates to true.

The attributes are explained as follows:

- *name*: The name of the attribute in context whose value is being tested.

The context is the defined as the HTTP request parameter, the JSP page context, or the current HTTP session.

<om:notexist />

Specification in taglib:

```
 <tag>
    <name>notexist</name>
    <tagclass>oracle.panama.studio.taglib.NotExistTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>A logical not-exists tag.</info>
    <attribute>
      <name>name</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>
```

Usage:

```
<om:notexist name= "attrName" >….</om:notexist>
```

<om:notexist> is a body tag, that is, its body content is evaluated and output as appropriate if the test condition evaluates to true.

The attributes are explained as follows:

- *name*: The name of the attribute in context whose value is being tested.

The context is defined as the HTTP request parameter, the JSP page context, or the current HTTP session.

<om:if />

Specification in taglib:

```
<tag>
   <name>if</name>
   <tagclass>oracle.panama.studio.taglib.IfTag</tagclass>
   <bodycontent>JSP</bodycontent>
   <info>A logical if tag.</info>
   <attribute>
     <name>name</name>
     <required>true</required>
     <rtexprvalue>true</rtexprvalue>
   </attribute>
   <attribute>
     <name>value</name>
     <required>true</required>
     <rtexprvalue>true</rtexprvalue>
   </attribute>
   <attribute>
     <name>op</name>
     <required>true</required>
     <rtexprvalue>false</rtexprvalue>
   </attribute>
</tag>
```

Usage:

```
<om:if name= “attrName” value= “attrValue”  op= “equal” >
   <om:then> ….</om:then>
  <om:elseif name=”attrName” value=”attrValue” op=”equal”>
    <om:then> …</om:then>
      <om:ese>
          <om:then> …</om:then>
    </om:else>
  </om:elseif>
</om:if>
```

<om:if> is used in combination with <om:elseif>, <om:else> and <om:then>.

If the application of the operator between the value of the given attribute in context and the value given evaluates to true, then the immediate child <om:then> tag's content are evaluated and output. Otherwise, the <om:elseif> or <om:else> that is an immediate child of the <om:if> is evaluated, and its contents are output as appropriate to them.

The attributes are explained as follows:

- *name*: The name of the attribute in context whose value is being tested.

The context is defined as the HTTP request parameter, the JSP page context, or the current HTTP session.

value: The value of the attribute being testing against.

<om:elseif>

Specification in taglib:

```
  <tag>
     <name>elseif</name>
     <tagclass>oracle.panama.studio.taglib.ElseIfTag</tagclass>
     <bodycontent>JSP</bodycontent>
     <info>A logical elseif tag, allowed only inside if or elseif.</info>
     <attribute>
       <name>name</name>
       <required>true</required>
       <rtexprvalue>true</rtexprvalue>
     </attribute>
     <attribute>
       <name>value</name>
       <required>true</required>
       <rtexprvalue>true</rtexprvalue>
     </attribute>
     <attribute>
       <name>op</name>
       <required>true</required>
       <rtexprvalue>false</rtexprvalue>
     </attribute>
  </tag>
```

Usage:

```
<om:if name= “attrName” value= “attrValue”  op= “equal” >
   <om:then> ….</om:then>
  <om:elseif name=”attrName” value=”attrValue” op=”equal”>
    <om:then> …</om:then>
      <om:ese>
          <om:then> …</om:then>
     </om:else>
  </om:elseif>
</om:if>
```

<om:elseif> is used in combination with <om:if>, <om:else> and <om:then>.

If the application of the operator between the value of the given attribute in context and the value given evaluates to true, then the immediate child <om:then> tag's content are evaluated and output Otherwise, the <om:else> that is the immediate child of the <om:elseif> is evaluated, and its contents are output as appropriate to it.

The attributes are explained as follows:

- *name*: The name of the attribute in context whose value is being tested.

The context is defined as the HTTP request parameter, the JSP page context, or the current HTTP session.

value: The value of the attribute being tested against.

<om:else />

Specification in taglib:

```
<tag>
   <name>else</name>
   <tagclass>oracle.panama.studio.taglib.ElseTag</tagclass>
   <bodycontent>JSP</bodycontent>
   <info>A logical else tag.</info>
</tag>
```

Usage:

```
<om:if name= “attrName” value= “attrValue”  op= “equal” >
   <om:then> ….</om:then>
  <om:elseif name=”attrName” value=”attrValue” op=”equal”>
    <om:then> …</om:then>
      <om:ese>
          <om:then> …</om:then>
     </om:else>
  </om:elseif>
</om:if>
```

<om:else> is used in combination with <om:elseif>, <om:else> and <om:then>.

When the parent <om:if> or <om:elseif> evaluates to false, then the content of the child <om:then> tag is evaluated and output as appropriate, otherwise, it does nothing.

<om:then />

Specification in taglib:

```
<tag>
   <name>then</name>
   <tagclass>oracle.panama.studio.taglib.ThenTag</tagclass>
   <bodycontent>JSP</bodycontent>
   <info>A logical else tag.</info>
</tag>
```

Usage:

<om:if name= “attrName” value= “attrValue” op= “equal” >

<om:then> ....</om:then>

<om:elseif name=”attrName” value=”attrValue” op=”equal”>

<om:then> ...</om:then>

<om:ese>

<om:then> ...</om:then>

</om:else>

</om:elseif>

</om:if>

`<om:then>` is used in combination with `<om:if>` , `<om:elseif>` and `<om:else>`.

If the parent `<om:if>` , `<om:elseif>` and `<om:else>` evaluates to true, then the contents of the `<om:then>` tag are evaluated and output as appropriate.

<om:iterate />

Specification in taglib:

```
<tag>
   <name>iterate</name>
   <tagclass>oracle.panama.studio.taglib.IterateTag</tagclass>
   <!-- teiclass>oracle.panama.studio.taglib.IterateTagTEI</teiclass -->
   <bodycontent>JSP</bodycontent>
   <info>An iteration tag.</info>
   <attribute>
     <name>name</name>
     <required>true</required>
```

```
      <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>
```

Usage:

```
<om:iterate name= “collectionName” >
…
</om:iterate>
```

<om:iterate> is used for iterating over a collection of bean objects. If an object with the given name is found in the context, and it is of Java Collection type, then the collection can be looped through, and each of the object inside the collection can be used as we loop through it. The body of the <om:iterate> tag will be output *n* times where *n* is the number of objects inside the collection.

<om:switch />

Specification in taglib:

```
<tag>
   <name>switch</name>
   <tagclass>oracle.panama.studio.taglib.SwitchTag</tagclass>
   <bodycontent>JSP</bodycontent>
   <info>A switch tag.</info>
   <attribute>
     <name>name</name>
     <required>true</required>
     <rtexprvalue>true</rtexprvalue>
   </attribute>
</tag>
```

Usage:

```
<om:switch name= “attrName”>
<om:case value= “value1”>
 …
</om:case>
<om:case value= “value2” >
…
</om:case>
…
<om:default>
…
</om:default>
</om:switch>
```

<om:switch> is used in combination with <om:case> and <om:default>, where in <om:switch> the name of the attribute being tested is specified, and inside <om:case> the value of the attribute being testing against is specified. In case of a match, the body of the <om:case> that satisfies the match is evaluated and output as appropriate, otherwise if the <om:default> is specified, then its body is evaluated and output instead.

<om:case />

Specification in taglib:

```
<tag>
   <name>case</name>
   <tagclass>oracle.panama.studio.taglib.CaseTag</tagclass>
   <bodycontent>JSP</bodycontent>
   <info>A case tag.</info>
   <attribute>
      <name>value</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
   </attribute>
</tag>
```

Usage:

See that of <om:switch >

<om:default />

```
Specification in taglib:
<tag>
   <name>default</name>
   <tagclass>oracle.panama.studio.taglib.DefaultTag</tagclass>
   <bodycontent>JSP</bodycontent>
   <info>A default tag. </info>
</tag>
```

Usage:

See that of <om:switch>

Part IV

Oracle9*i*AS Wireless Modules

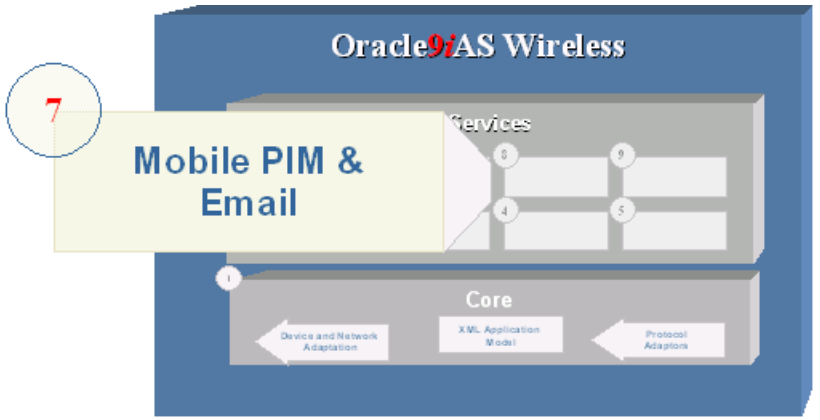
Part IV contains information about Oracle9*i*AS Wireless modules.

18

Mobile PIM and eMail

Each section of the chapter describes a module and its configuration. The sections of this chapter include:

Figure 18–1 The Mobile PIM and Email



18.1 Mobile PIM and eMail Overview

Oracle9*i* Application Server Personal Information Management (PIM) Service enables customers to integrate corporate email, directory, address book, calendaring and instant messaging applications into their mobile enterprise portals.

Each of these applications is built as a module that can be called either directly by mobile users from their devices, or by other applications. These mobile PIM and email applications are fully integrated within one another, enabling a user to access such features as an address book-based recipient selection or a directory when composing email messages.

Oracle9*i* Application Server customers can leverage the Personal Information Management Service modules into their own or third-party applications to add communication features to these services, to retrieve corporate directory information, or to add and manage appointments for users such as travel or dining reservations.

18.1.1 Mobile Email

The email module enables users to access their email messages from any mobile device. Mobile Email integrates with any IMAP or POP3 server (including Microsoft Exchange and Lotus Domino servers).

18.1.1.1 Configuring an Email Service

All the necessary **.jar** files are shipped with Oracle9*i*AS Wireless. The are no scripts.

Table 18–1 Email Module Parameters

Parameter	Description	Default Value	Example
ORACLE_SERVICES_PIM_ MAIL_PROTOCOL	The mail protocol.	pop3	imap, pop3
ORACLE_SERVICES_PIM_ MAIL_SERVER_NAME	Full hostname of the Mail server.	localhost	mailserver.mycompany.com
ORACLE_SERVICES_PIM_ MAIL_SERVER_PORT	Port number of the Mail server on the host.	110	Typically 143 for imap and 110 for pop3.
ORACLE_SERVICES_PIM_ MAIL_SMTP_SERVER_NAME	Full hostname of the SMTP server.	localhost	mysmtp.mycomp.com
ORACLE_SERVICES_PIM_ MAIL_SMTP_SERVER_PORT	Port number of the SMTP server on the host.	25	25, 45
ORACLE_SERVICES_PIM_ MAIL_SMTP_SERVER_LOGIN	The login name of the SMTP server. If login for SMTP is not required, then this value should be null or empty # string (""), otherwise, the value will be used for SMTP login. The format is <login name>@<domain>. If domain is not specified here, the value of the property ORACLE_SERVICES_PIM_MAIL_ AUTODOMAIN will be appended. If that is not specified also, the property ORACLE_ SERVICES_PIM_MAIL_DEFAULT_ EMAILDOMAIN will be used.	none	john.doe@mycomp.com
ORACLE_SERVICES_PIM_ MAIL_AUTODOMAIN	When composing email, if email address does not have a domain, this domain will be automatically appended.	localhost	mycomp.com
ORACLE_SERVICES_PIM_ MAIL_FOLDER_INBOX	The name of the Inbox folder on the mail server.	INBOX	INBOX, Inbox
ORACLE_SERVICES_PIM_ MAIL_FOLDER_SENT	The name of the Sent folder on the mail server.	Sent	Sent, SentItems
ORACLE_SERVICES_PIM_ MAIL_DEFAULT_ EMAILDOMAIN	The default email domain for sending emails. When sending emails, if the email domain is not specified, then the default email domain will be appended to the username. For example, if the default domain is oraclemobile.com, and a user sends an email to the user "john.doe", the email will be sent to john.doe@mycomp.com	localhost	mycomp.com

Table 18–1 Email Module Parameters

Parameter	Description	Default Value	Example
ORACLE_SERVICES_PIM_MAIL_MSGFETCH_SETSIZE	The maximum number of messages fetched for one request. This is the maximum number of messages return for one request. For example, if a folder has 300 messages, only the first 200 will be returned at the first time.	200	200, 300
ORACLE_SERVICES_PIM_MAIL_SERVER_CONNECT_TIMEOUT	The connection timeout in milliseconds.	2000	2000, 3000
ORACLE_SERVICES_PIM_MAIL_CONFIG_CLASS	The configuration class. Users can provide their own configuration classes, enabling them to perform advanced configuration such as selecting a mail server host name based on runtime information.	oracle.panama.module.pim.mail.util.Config	oracle.panama.module.pim.mail.util.Config mycomp.mail.config.MailConfig
ORACLE_SERVICES_PIM_MAIL_TEMP_DIR	The temporary directory for downloading a message or attachments.	/temp	/home/9iasuser/temp, D:\9iasuser\temp
ORACLE_SERVICES_PIM_MAIL_AUDIO_TMP_DIR	The temporary directory that stores the temporary audio files. This directory holds the **.wav** files that are generated when a user replies to an email in the voice version of the application. This directory should be web accessible.	/modules/modules-web/pim/mail/audiotemp	/home/9iasuser/iaswv20/modules/modules-web/pim/mail/audiotemp
ORACLE_SERVICES_PIM_MAIL_AUDIO_TMP_URL	The url for the temporary audio files directory.	http://localhost/modules/pim/mail/audiotemp	http://localhost/modules/pim/mail/audiotemp
ORACLE_SERVICES_PIM_MAIL_MESSAGE_ENCODING	The encoding that is used while sending an email. If this field is empty, then the default system encoding is used.	UTF-8	UTF-8, Shift_JIS, Big5

18.1.1.2 Linking to a Email Service

You can link to an email service using the following virtual URL:

omp://oracle/services/pim/mail

Input Call Parameters

The input call parameters of the email module include the following:

action

The action the email module should perform. This is a mandatory input parameter.

Table 18–2 Input Parameters for Action

Valid Value	Description	Requirement
messageto	Send an email message.	Requires `mailto`.
messagecc	CC an email message.	Requires `mailto`.
`sendasattachment`	Send an attachment	Requires `attach`.

mailto

The email address to which the message is sent. This is an optional input parameter. The value must be a string. For example:

- mailto=oraclemobile@oracle.com
- mailto=john.smith@mycompany.com

attach

The fully-qualified path of the local file that is sent as an attachment to the email. the value must be a string. For example:

- attach=/home/9iasuser/temp/presentation.ppt
- attach=D:\9iasuser\temp\instructions.txt

Output Parameters (Examples)

To send an email to Scott Tiger, you configure the action and mailto parameters as follows:

- action=messageto
- mailto=scott.tiger@oracle.com

To send the picture (a **.jpeg**)of your new home, you configure the action and attach parameters as follows:

- action=sendasattachment
- attach=/private/9iasuser/temp/my1MilDolHome.jpg

18.1.2 Mobile Directory

The Mobile Directory module enables users to access LDAP directory servers from any mobile device. The directory module is integrated with the mobile email module, enabling users to browse their corporate directory and then send an email to a particular contact, or to compose a recipient list from the directory.

18.1.2.1 Configuring the Mobile Directory

All of the required **.jar** files are included with Oracle9*i*AS Wireless. This module requires no scripts.

The Mobile directory includes the following configuration parameters:

Table 18–3 Configuration Parameters for the Mobile Directory

Parameters	Description	Default Value	Examples
ORACLE_SERVICES_PIM_DIRECTORY_HOST	The LDAP server hostname.	localhost Valid value: myldapserver.cycompany.com	
ORACLE_SERVICES_PIM_DIRECTORY_PORT	The LDAP server port number	389	389,345
ORACLE_SERVICES_PIM_DIRECTORY_LOGIN	Is Login required or not?	The valid value is 'false'.	true, false
ORACLE_SERVICES_PIM_DIRECTORY_USERNAME	If a LDAP login is required, then a default username and password can be specified. Set ORACLE_SERVICES_PIM_DIRECTORY_LOGIN to true.	my user	User 1234
ORACLE_SERVICES_PIM_DIRECTORY_PASSWORD	If a LDAP login is required then a default username and password can be specified. Set ORACLE_SERVICES_PIM_DIRECTORY_LOGIN to true.	1234	55 hello 27478
ORACLE_SERVICES_PIM_DIRECTORY_MAX_RESULT_COUNT	Maximum results returned from a query.	1000	1000, 1200

Parameters	Description	Default Value	Examples
ORACLE_SERVICES_PIM_DIRECTOR_QUERY_NAMES	The list of IDs for each type of query to be defined. For each one defined, other parameters with the ID need must be created as specified below. The parameters with the ID "BYNAME" must be created for each query.	None	BYNAME, BYEMAIL
ORACLE_SERVICES_PIM_DIRECTORY_MAX_REC_PAGE	The maximum results displayed on a screen.	9	9, 10
ORACLE_SERVICES_PIM_DIRECTORY_MERGE_RESULTS	Whether to display other attributes returned by a query but not specified in the filter.		true, false
ORACLE_SERVICES_PIM_DIRECTORY_HOTLINK_NAMES	The list of hotlink names. You must define corresponding parameters for each hotlink you define. The parameters with the ID "MANAGER" are required for each hotlink.		MANAGER
ORACLE_SERVICES_PIM_DIRECTORY_QUERY_BYNAME_DISPLAY	The display name for this query, which is displayed to the user.		Search by Name.
ORACLE_SERVICES_PIM_DIRECTORY_QUERY_BYNAME_SEARCHBASE	Identifies the location from which the search begins.		Examples (dc=oracle,dc=com)
ORACLE_SERVICES_PIM_DIRECTORY_QUERY_BYNAME_SEARCHSCOPE	Identifies the scope of the search.		BASE, ONE, SUBTREE
ORACLE_SERVICES_PIM_DIRECTORY_QUERY_BYNAME_VISIBLE	Whether a query is visible as a choice to the user.		true, false
ORACLE_SERVICES_PIM_DIRECTORY_QUERY_BYNAME_FILTER	The filter attributes for the query.		cn, sn, email
ORACLE_SERVICES_PIM_DIRECTORY_QUERY_FILTER_EXPRESSION	The Filter expression for the query, which is RFC 2254 - compliant.		(!(cn=Scott Tiger)), cn=*?*

Parameters	Description	Default Value	Examples
ORACLE_SERVICES_PIM_DIRECTORY_ QUERY_BYNAME_FILTER_DISPLAY	A prompt for the input values of the filter.		Enter a name to search:, Enter an area to search
ORACLE_SERVICES_PIM_DIRECTORY_ QUERY_BYNAME_RESULTLIST	The list of attributes that you want from the query.		
ORACLE_SERVICES_PIM_DIRECTORY_ QUERY_BYNAME_RESULTLIST_DISPLAY	The display names for the attributes that are retrieved.	.	First Name, Last Name, Email Address
ORACLE_SERVICES_PIM_DIRECTORY_ QUERY_BYNAME_RESULTLIST_TYPE	The type of link (made from the results).		display, phone, fax, email, link, SMS
ORACLE_SERVICES_PIM_DIRECTORY_ QUERY_BYNAME_RESULTLIST_HOTLINK	Specifies if the result attribute is a hotlink. Set to false if the result attribute is not a hotlink; otherwise, any other value will suffice.		false, MANAGER, GROUP
ORACLE_SERVICES_PIM_DIRECTORY_ QUERY_BYNAME_HOTLINK_NAMES	The names for the hotlink of an attribute. If the corresponding attribute in the list is not a hotlink, enter "nope"	.	nope, MANAGER,
ORACLE_SERVICES_PIM_DIRECTORY_ HOTLINK_MANAGER_QUERY	The query name used by the hotlink.		
ORACLE_SERVICES_PIM_DIRECTORY_ HOTLINK_MANAGER_COLUMN_REFER	The column values to use from the current result set.		cn, sn
ORACLE_SERVICES_PIM_DIRECTORY_ HOTLINK_MANAGER_COLUMN_BIND	The attribute names to bind the referenced values.		cn, sn
ORACLE_SERVICES_PIM_DIRECTORY_ HOTLINK_MANAGER_COLUMN_DISPLAY	The attribute values that are displayed. These values are delimited by a space in the display.		givenname, sn

Parameters	Description	Default Value	Examples
ORACLE_SERVICES_PIM_DIRECTORY_ QUERY_BYNAME_RESULTLIST_ SHORTLIST	Comma-delimited series of tokens, either "true" or "false", and is of the same length as the parameter ORACLE_SERVICES_ PIM_DIRECTORY_ QUERY_BYNAME_ RESULTLIST. The value of the token specifies whether that particular member of the result list will be displayed in the short list mode (value = true) or if the member of the result list will only be displayed in the detailed view (value = false).		

18.1.2.2 Linking to the Directory Module

You can link to the directory module through the following virtual URL:

omp://oracle/services/pim/directory

You can configure each element of this service; clicking any field after getting the details of a result returns the field value to the caller as the parameter `mailto`.

Output Parameters

The output parameters for the mobile directory service include the following:

mailto

The value of the field that the user clicks. For example:

- mailto=oraclemobile@oracle.com
- mailto=John
- mailto=Smith
- mailto=(650)999-9999

There are no restrictions for this parameter.

Examples

To return a first name, configure the mailto parameter as follows:

mailto=john

To return an email address, cofeature the mailto parameter as follows:

mailto=john.smith@mycompany.com

18.1.3 Mobile Address Book

The Mobile address book enables users to manage their own address books and contacts as well as enabling call functions from wireless phones. The mobile address book integrates with the Mobile Email Module to allow users to compose a messages' recipient list from their address book.

Once you find a contact, you can also edit the contact information or delete a contact. While deleting, nothing is returned to the caller.

18.1.3.1 Configuring the Mobile Address Book

This service implements two distinct modes, both with the same user experience but different back-ends. In its Oracle9*i*AS Wireless 2.0 mode, it does not require any third-party software and uses Oracle9*i*AS Wireless 2.0 storage. In its Microsoft Exchange Mode, it fully integrates with a Microsoft Exchange server to mobile enable Exchange users

Required Software

The mobile address book requires the following third-party software:

Table 18-1 Required Third-Party Software for the Mobile Address Book

Name	Instructions	Version(s)
MS Exchange	[Exchange Mode] Install the Microsoft Exchange Server.	5.5
MS IIS	[Exchange Mode] Install the Microsoft Internet Information Server.	4.0
MS CDO	[Exchange Mode] Collaborative Data Objects. Available with Exchange SDK. The cdo.dll library must be installed on the IIS Server.	1.2.1

Table 18-1 Required Third-Party Software for the Mobile Address Book

Name	Instructions	Version(s)
Oracle9*i*AS Wireless 2.0 Exchange ASP (Shipped with Oracle9*i*AS Wireless 2.0)	[Exchange Mode] Create any directory on the IIS server. Copy all the files in $ORACLE_HOME/iaswv20/wireless/j2ee/applications/modules/modules-web/pim/addressbook/asp/ for a Solaris install, %ORACLE_HOME%\iaswv20\wireless\j2ee\applications\modules\modules-web\pim\addressbook\asp\ for an NT install to the just created directory on the IIS server. Invoke the properties dialog box for this directory. Choose the Directory tab. Click the "Edit" button for the "Anonymous Access and Authentication Control". Set as follows: ■ Allow Anonymous Access - unchecked ■ Basic Authentication - checked ■ Windows NT Challenge/Response - checked	

Configuration Parameters

The mobile address book includes the following parameters:

- ORACLE_SERVICES_PIM_ADDRESSBOOK_CLASS
  - Description: The service implementation class. Determines whether the Oracle9*i*AS or Exchange mode is used.
  - Default Value: oracle.panama.module.pim.addressbook.omaddressbook.OMAddressBook
  - Valid Values: oracle.panama.module.pim.addressbook.omaddressbook.OMAddressBook, oracle.panama.module.pim.addressbook.exchange.ExchangeAddressBook.
- ORACLE_SERVICES_PIM_ADDRESSBOOK_SERVER_NAME
  - Description: If Exchange mode is selected, designates the Exchange server host name or IP address. It should not contain a value in the Wireless 2.0 mode.
  - Default Value: localhost
  - Valid Values: exchange.mycompany.com, email-serv.mycompany.com.
- ORACLE_SERVICES_PIM_ADDRESSBOOK_SERVER_DATA_WEB_LINK

- Description: If Exchange mode is selected, this is the URL pointing to the Oracle9*i*AS Wireless Exchange Address Book ASP (AddressBook.asp) running on IIS. It should have no value in the Wireless 2.0 mode.
- Default Value: None
- Examples: http://iis.mycompany.com/Oracle9iASW20/exchange/AddressBook.asp

- ORACLE_SERVICES_PIM_ADDRESSBOOK_DOMAIN
  - Description: The domain attached to the user account information. This is a free form field. If the Oracle9*i*AS Wireless instance supports multiple address book servers, then these address books must not have the same domain. Different PIM services with the same domain share the user account information.
  - Default Value: LOCALDOMAIN
  - Examples: LOCALDOMAIN for the Oracle9*i*AS Wireless Mode, ExchangeDomain for Exchange.

18.1.3.2 Linking to the Mobile Address Book

You can link to the mobile address book using the following virtual URL:

omp://oracle/services/pim/addressbook

The mobile address book includes the following call parameters:

Table 18–4 Input Call Parameters of the Mobile Address Book

Parameter Name	Mandatory	Description	Valid Value
screen	No	The function performed by the Addressbook.	0 (Displays the list of contacts); 51 (Makes the Addressbook service add the contact with the provided data to this database of contacts. This parameter requires SERIALIZED_CONTACT if the value is 51.)
srchstr	No	Makes the Addressbook perform a search for the specified string among all of the contacts.	The string to search for. Requires screen := {0 empty}

SERIALIZED_CONTACT

The SERIALIZED_CONTACT group contains the parameters for each element of a contact, such as contact name, contact work phone, and contact work address. The elements described in this optional group are returned when the user clicks done in the screen with a contact detail.

The SERIALIZED_CONTACT group includes the following parameters:

Table 18–5 Parameters of the Serialized Contact Group for Addressbook

Parameter Name	Mandatory	Description	Valid Value
NAME	Yes	The name of this contact.	For example, NAME=John Smith.
WORKPH	No	The work phone number of this contact. Restriction: white-spaces, special characters, are encoded.	WORKPH=650-123-4567
MOBILEPH	No	The mobile phone number of this contact.	
HOMEPH	No	The home phone of this contact. Restriction: white-spaces, special characters, are encoded.	
WORKFAX	No	The business fax number of this contact. Restriction: white-spaces, special characters, are encoded.	
EMAIL1	No	The email (or the first email) address of this contact. Restriction: white-spaces, special characters, are encoded.	An email address, for example, EMAIL1=scott.tiger@oralce.com
EMAIL2	No	The second email address of this contact. Restriction: white-spaces, special characters, are encoded.	An email address, for example, EMAIL2=scott.tiger@homemail.com
WADDRLINE1	No	The first (or only) line of the Work address of this contact. Restriction: white-spaces, special characters, are encoded.	The first line of a street address. For example: WADDRLINE1=123 Main Street

Parameter Name	Mandatory	Description	Valid Value
WADDRCITY	No	The city or Work address of this contact. Restriction: white-spaces, special characters, are encoded.	A city; for example, WADDRCITY = Boston
WADDRSTATE	No	The state (or federal region) of the WORK address of this contact. Restriction: white-spaces, special characters, are encoded.	A state (or federal region); for example, WADDRSTATE = CA WADDRSTATE = Massa chusetts
WADDRZIP	No	The ZIP or postal code of the work address of this contact.	A ZIP or postal code. For example, WADDRZIP=02142 WADDRZIP=D-80333
WADDRCOUNTRY	No	The country of the work address of this contact.	The name of a country, for example: WADDRCOUNTRY=U.S.A.
HADDRLINE1	No	The first (or only) street line of the home address of this contact. Restriction: white-spaces, special characters, are encoded.	The first line of a street address, for example: HADDRLINE1 = 2901 Armstrong Dr.
HADDRCITY	No	The city of the home address of the person in the contact.	The name of a city, for example: HADDRCITY=Boston
HADDRSTATE	No	The state (or federal region) of the home address of the person in this contact.	The full name or abbreviation of the state. For example: HADDRSTATE=Massachusetts HADDRSTATE=CA
HADDRZIP	No	The ZIP or postal code of this contact.	The ZIP or postal code. For example: HADDRZIP=90210 HADDRZIP=D-80333
HADDRCOUNTRY	No	The country of the home address of this contact.	A name of a country, for example: HADDRCOUNTRY=U.S.A.
NOTES	Yes	Text notes describing this contact. Restriction: white-spaces, special characters, are encoded.	A short description of the person in the contact, for example: NOTES=This the chief-of-staff in CCC Co.

Output Parameters

The output parameters for the Addressbook include the following:

Table 18–6 Output Parameters of the Addressbook

Parameter Name	Mandatory	Description
mailto	No.	An email address of a contact. This must be an email address. For example: mailto=scott.tiger@orcale.com

smPhone

smphone is a phone number of a contact, returned with additional parameters used by the Short Messaging module (usually when the user clicks on a phone number in the Addressbook).

This group includes the following parameters:

Table 18–7 Parameters of smPhone

Parameter Name	Mandatory	Description	Valid Value
type	Yes	The type of short messaging service desired.	VOICE, FAX
destinationAddress	Yes	The recipient number of address for the short message (usually a phone number).	A phone number, for example: destinationAddress=650-555-5000.

faxNumber

faxNumber is the fax number of a contact, returned with additional parameters used by the FAX or Short-Messaging modules (usually when the user clicks on a fax number in the Addressbook). The faxNumber group includes the following parameters:

Table 18–8 Parameters of faxnumber

Parameter Name	Mandatory	Description	Valid Value
type	Yes	The type of short messaging service needed.	FAX
destinationAddress	Yes	The fax number of the recipient used in the short messaging module.	A fax number, for example, destinationAddress=650-123-4567

Parameter Name	Mandatory	Description	Valid Value
FAXTODO	Yes	The function that the fax module perform.	NEWFAX
RNAME	Yes	The name of the recipient of the fax.	A name, for example: RNAME=Scott Tiger
RPHONE	Yes	The phone number of the recipient of the fax.	a phone number, for example: RPHONE=650-555-5000.
RFAX	Yes	The fax number to which the fax is sent.	a fax number, for example, RFAX=650-555-1234

SERIALIZED_CONTACT

The SERIALIZED_CONTACT group contains the parameters for each element of a contact, such as contact name, contact work phone, and contact work address. The elements described in this optional group are returned when the user clicks done in the screen with a contact detail.

The parameters of the SERIALIZED CONTACT group are as follows:

Table 18–9 Parameters of the SERIALIZED_CONTACT Group

Parameter Name	Mandatory	Description	Valid Value
NAME	Yes	The name of this contact. Restriction: the white-spaces, special characters, are encoded.	A name. For example, NAME=John Smith.
WORKPH	No	The work phone number of this contact. Restriction: the white-spaces, special characters, are encoded.	A phone number, for example: WORKPH=650-123-4567
HOMEPH	No	The home phone number of this contact. Restriction: the white-spaces, special characters, are encoded.	A phone number, for example: HOMEPH=650-555-5000
MOBILEPH	No	The mobile phone number of this contact. Restriction: the white-spaces, special characters, are encoded.	A phone number, for example: MOBILEPH=650-555-5000

Parameter Name	Mandatory	Description	Valid Value
WORKFAX	No	The business fax number of this contact. Restriction: the white-spaces, special characters, are encoded.	Example: WORKFAX=
EMAIL1	No	The e-mail (or the first email) address of this contact. Restriction: the white-spaces, special characters, are encoded.	An email address, for example, EMAIL1=scott.tiger@oralce.com
EMAIL2	No	The second email address of this contact. Restriction: the white-spaces, special characters, are encoded.	An email address, for example, EMAIL2=scott.tiger@homemail.com
WADDRLINE1	No	The first (or only) line of the Work address of this contact. Restriction: the white-spaces, special characters, are encoded.	The first line of a street address. For example: WADDRLINE1=123 Main Street
WADDRCITY	No	The city or work address of this contact. Restriction: the white-spaces, special characters, are encoded.	A city; for example, WADDRCITY = Boston
WADDRSTATE	No	The state (or federal region) of the WORK address of this contact. Restriction: the white-spaces, special characters, are encoded.	A state (or federal region); for example, WADDRSTATE = CA WADDRSTATE = Massachusetts
WADDRZIP	No	The ZIP or postal code of the work address for this contact.	A ZIP or postal code. For example, WADDRZIP=02142 WADDRZIP=D-80333
WADDRCOUNTRY	No	The country of the work address of this contact.	The name of a country, for example: WADDRCOUNTRY=U.S.A.
HADDRLINE1	No	The first (or only) street line of the home address of this contact. Restriction: the white-spaces, special characters, are encoded.	The first line of a street address, for example: HADDRLINE1 = 2901 Armstrong Dr.

Parameter Name	Mandatory	Description	Valid Value
HADDRCITY	No	The city of the home address of the person in the contact.	The name of a city, for example: HADDRCITY=Boston
HADDRSTATE	No	The state (or federal region) of the home address of the person in this contact.	The full name or abbreviation of the state. For example: HADDRSTATE=Massachusetts HADDRSTATE=CA
HADDRZIP	No	The ZIP or postal code of this contact.	The ZIP or postal code. For example: HADDRZIP=90210 HADDRZIP=D-80333
HADDRCOUNTRY	No	The country of the home address of this contact.	A name of a country, for example: HADDRCOUNTRY=U.S.A.
NOTES	Yes	Text notes describing the person this contact. Restriction: the white-spaces, special characters, are encoded.	A short description of the person in the contact, for example: NOTES=This the chief-of-staff in CCC Co.

18.1.4 Calendar

The calendar enables users to manage their schedule and tasks via mobile access to calendaring servers, such as Microsoft Exchange and Lotus Domino.

18.1.4.1 Configuring the Calendar Module

This service implements two distinct modes, both with the same user experience but with different back-ends. In its Lotus Domino mode, it fully integrates with a Lotus Domino server to mobile enable Domino users. In its Microsoft Exchange Mode, it fully integrates with a Microsoft Exchange server to mobile enable Exchange users.

Required Third-Party Software

The calendar module requires the following third-party software:

Table 18–10 Software Requirements for the Calendar Module

Name	Instruction	Version(s)
MS Exchange	Exchange mode: Install the Microsoft Exchange Server.	5.5
MS IIS	Exchange mode: Install the Microsoft Internet Information Server.	4.0
MS CDO	Exchange mode: Collaborative Data Objects. Available with Exchange SDK. The cdo.dll library must be installed on the IIS Server.	1.2.1
Oracle9*i*AS Wireless (Exchange ASP Shipped with Oracle9*i*AS Wireless 2.0)	Exchange mode: Create any directory on the IIS server. Copy all the files in $ORACLE_HOME/iaswv20/wireless/j2ee/applications/modules/modules-web/pim/calendar/asp/ for a Solaris install, %ORACLE_HOME%\iaswv20\wireless\j2ee\applications\modules\modules-web\pim\calendar\asp\ for an NT install to just created directory on the IIS server. Invoke the properties dialog box for this directory. Choose the Directory tab. Click the "Edit" button for the "Anonymous Access and Authentication Control", make sure the following check boxes are set as follows: ■ Allow Anonymous Access - unchecked ■ Basic Authentication - checked ■ Windows NT Challenge/Response-checked	

Name	Instruction	Version(s)
MS Exchange	Exchange mode: Install the Microsoft Exchange Server.	5.5
Lotus Java SDK	Domino Mode: Install the Lotus Domino Toolkit for Java / CORBA, and add NCSO.jar to the classpath for Oracle9*i*AS Wireless 2.0. Installing the toolkit creates "DTJava" directory on the file system. Copy the DTJava / lib / NCSO.jar to $ORACLE_HOME / wireless / lib on Solaris, to %ORACLE_ HOME% \ wireless \ lib on NT.Examples of ORACLE_HOME values: Solaris: ORACLE_HOME= / u01 / iaswv20NT: ORACLE_HOME=d: \ oracle \ iaswv20 Ensure you download "Lotus Domino Toolkit for Java / CORBA Release 5.0.8 Update" or "Lotus Domino Toolkit for Java / CORBA Release 5.0.5 Update Shipping". Do not use version 2.x toolkit. On the Domino server, the server tasks HTTP and DIIOP must be running. Ensure that the Domino server notes.ini file contains the following line: `ServerTasks=<other tasks>,http,diiop`	R5

Configuration Parameters

The Calendar module includes the following configuration parameters:

- ORACLE_SERVICES_PIM_CALENDAR_CLASS
  - Description: The service implementation class, which determines whether the Domino or Exchange mode is used.
  - Default Value: None
  - Valid Values: oracle.panama.module.pim.calendar.domino.DominoCalendarService, oracle.panama.module.pim.calendar.exchange.ExchangeCalendarService.
- ORACLE_SERVICES_PIM_CALENDAR_SERVER_NAME

- Description: Designates the Exchange server or Domino server host name.
- Default Value: None
- Examples: exchange.mycompany.com, domino-server.mycompany.com

- ORACLE_SERVICES_PIM_CALENDAR_SERVER_DATA_WEB_LINK
  - Description: If the Exchange mode is selected, this is the URL pointing to the Oracle9*i*AS Wireless 2.0 Exchange Calendar ASP (Calendar.asp) running on IIS.
  - Default Value: None
  - Examples: http://iis.mycompany.com/Oracle9iASW20/exchange/Calendar.asp
- ORACLE_SERVICES_PIM_CALENDAR_DOMAIN
  - Description: The domain is attached to the user account information. This is a free-form field. If the Wireless instance Supports multiple calendar servers, then these must not have the same domain. Different PIM services with the same domain share the user account information.
  - Default Value: None
  - Examples: DominoDomain for the Domino Mode, ExchangeDomain for Exchange.

18.1.4.2 Linking to the Calendar Module

You can link to the calendar module using the following virtual URL:

omp://oracle/services/pim/calendar

Input Call Parameters for the Calendar Module

The input call parameters of the calendar module include the `getApptDetails` group. This optional group includes the following input call parameters:

Table 18–11 Parameters of ID

Parameter Name	Mandatory	Description	Valid Value
ID	Yes	The input ID required to retrieve appointment details.	A string. For example, ID=1324.

Table 18–12 Parameters of addAppt

Parameter Name	Mandatory	Description	Valid Value
TITLE	Yes	The title of the appointment.	A string. For example, TITLE=Dinner at Joe's.
DATE	Yes	The date of the appointment.	A string. For example, DATE=December 31, 2001
TIME	Yes	The time of the appointment.	A string. For example, TIME= 8:00 p.m.
DURATION	Yes	The duration of the appointment.	A string. For example, DURATION=1 hour.
NOTES	Yes	The notes for the appointment.	A string. For example, NOTES=Remember the brief.
TYPE	Yes	The type of appointment, either personal or business.	A string. For example, TYPE=Business.
LOCATION	Yes	The location of an appointment.	A string. For example, LOCATION=Home.
REMIND	Yes	The time interval before the event reminder occurs.	A string. For example, REMIND=1 hour.
SHARING	Yes	A flag that enables or disables the sharing of an appointment. If True, the appointment is shared; if FALSE, then the appointment is not shared.	For example, SHARING=TRUE.

The Calendar module also includes the `deleteAppt` group. The `deleteAppt` group includes the following parameter:

Table 18–13 Parameter of deleteAppt

Parameter Name	Mandatory	Description	Valid Value
ID	Yes	The input ID required to select an appointment.	A string. For example, ID=1324.

Output Parameters of the Calendar Module

The calendar module includes the following output parameters:

The output parameters of the calendar module include the getApptDetailsRresponse group. This optional group contains the following parameters:

Table 18–14 The Output Parameters of the getApptDetailsResponse Group

Parameter Name	Mandatory	Description	Valid Value
TITLE	Yes	The title of the appointment.	A string. For example, TITLE=Dinner at Joe's.
DATE	Yes	The date of the appointment.	A string. For example, DATE=December 31, 2001
TIME	Yes	The time of the appointment.	A string. For example, TIME= 8:00 p.m.
DURATION	Yes	The duration of the appointment.	A string. For example, DURATION=1 hour.
NOTES	Yes	The notes for the appointment.	A string. For example, NOTES=Remember the brief.
TYPE	Yes	The type of appointment, either personal or business.	A string. For example, TYPE=Business.
LOCATION	Yes	The location of an appointment.	A string. For example, LOCATION=Home.
REMIND	Yes	The time interval before the event reminder occurs.	A string. For example, REMIND=1 hour.
SHARING	Yes	A flag that enables or disables the sharing of an appointment. If True, the appointment is shared; if FALSE, then the appointment is not shared.	For example, SHARING=TRUE.

apptResponse

The apptResponse group (an optional group) includes the following parameters:

Table 18–15 The Output Parameters of the addApptResponse Group

Parameter Name	Mandatory	Description	Valid Value
TITLE	Yes	The title of the appointment.	A string. For example, TITLE=Dinner at Joe's.
DATE	Yes	The date of the appointment.	A string. For example, DATE=December 31, 2001
TIME	Yes	The time of the appointment.	A string. For example, TIME= 8:00 p.m.
DURATION	Yes	The duration of the appointment.	A string. For example, DURATION=1 hour.
NOTES	Yes	The notes for the appointment.	A string. For example, NOTES=Remember the brief.

Parameter Name	Mandatory	Description	Valid Value
TYPE	Yes	The type of appointment, either personal or business.	A string. For example, TYPE=Business.
LOCATION	Yes	The location of an appointment.	A string. For example, LOCATION=Home.
REMIND	Yes	The time interval before the event reminder occurs.	A string. For example, REMIND=1 hour.
SHARING	Yes	A flag that enables or disables the sharing of an appointment. If True, the appointment is shared; if FALSE, then the appointment is not shared.	For example, SHARING=TRUE.

deleteApptResponse

The deleteApptResponse group (an optional group) includes the following parameters:

Table 18–16 Parameters of the deleteApptResponse group

Parameter Name	Mandatory	Description	Valid Value
TITLE	Yes	The title of the appointment.	A string. For example, TITLE=Dinner at Joe's.
DATE	Yes	The date of the appointment.	A string. For example, DATE=December 31, 2001
TIME	Yes	The time of the appointment.	A string. For example, TIME= 8:00 p.m.
DURATION	Yes	The duration of the appointment.	A string. For example, DURATION=1 hour.
NOTES	Yes	The notes for the appointment.	A string. For example, NOTES=Remember the brief.
TYPE	Yes	The type of appointment, either personal or business.	A string. For example, TYPE=Business.
LOCATION	Yes	The location of an appointment.	A string. For example, LOCATION=Home.
REMIND	Yes	The time interval before the event reminder occurs.	A string. For example, REMIND=1 hour.

Parameter Name	Mandatory	Description	Valid Value
SHARING	Yes	A flag that enables or disables the sharing of an appointment. If True, the the appointment is shared; if FALSE, then the appointment is not shared.	For example, SHARING=TRUE.

18.1.5 Instant Messaging

The instant messaging module provides presence management, enabling employees to exchange instant messages from their mobile devices. Integrated with Jabber Instant Messaging server and the MSN and Yahoo networks.

18.1.5.1 Configuring the Instant Messaging Module

The instant messaging module, which uses the Jabberbeans classes to connect to a Jabber Instant Messaging Server, requires the following third-party software.

Table 18–17 Third-Party Software for the Instant Messaging Module

Name	Instructions	Version(s)
Jabber Beans	Copy the latest **jabberbeans.jar** to $ORACLE_HOME/wireless/lib on Solaris, to %ORACLE_HOME%\wireless\lib on NT.Examples of ORACLE_HOME values: Solaris: ORACLE_HOME=/u01/iaswv20NT: ORACLE_HOME=d:\oracle\iaswv20	0.9.0-pre4
Jabber Server	Follow the Jabber server's installation guide.	1.4.1
Yahoo Transport Gateway	Optional. Follow the Jabber server's installation guide	0.8.0
MSN Transport Gateway	Optional. Follow the Jabber server's installation guide.	1.1.0

This module does not require scripts.

Configuration Parameters of the Instant Messaging Module

The instant messaging module includes the following configuration parameters:

Table 18–18 Instant Messaging Module Parameters

Parameter	Description	Default Value	Valid Values
ORACLE_SERVICES_ PIM_IM_PROXY_SET	Determines whether the service uses an HTTP proxy to access the Jabber Server.	false	true, false
ORACLE_SERVICES_ PIM_IM_PROXY_HOST	The host name of the HTTP proxy, if any, used to connect to the Jabber Server.	none	www-proxy.mycompany.com, http-proxy.mycompany.com
ORACLE_SERVICES_ PIM_IM_PROXY_PORT	The port number of the HTTP proxy, if any, used to connect to the Jabber Server.	80	Any valid TCP port number on which the HTTP proxy is listening (8080, etc).
ORACLE_SERVICES_ PIM_IM_SERVER_NAME	The host name of the machine on which the Jabber Server is configured.	localhost	jabber.mycompany.com, corporate-jabber.mycompany.com

ORACLE_SERVICES_ PIM_IM_USER_ DIRECTORY	The name of the user directory service configured on the Jabber Server accessed by the service.	None	Jud.jabber.mycompany.com
			
			directory.jabber.mycompany.com.
			
			This name should be set to the value of the jid attribute of the service element of type jud in the active Jabber server's **jabber.xml** configuration file:
			
			`<service type="jud" jid="jud.jabber.mycompany.com" name="Dir">`
			
			`<ns>jabber:iq:search</ns>`
			
			`<ns>jabber:iq:register</ns>`
			
			`</service>`

Table 18–18 Instant Messaging Module Parameters

Parameter	Description	Default Value	Valid Values

ORACLE_SERVICES_ PIM_IM_YAHOO	The Yahoo! Instant Messaging transport, if any, configured on the Jabber Server used by the service.	None	yahoo.jabber.mycompany.com
			yahoo-im.jabber.mycompany.com
			This name should be set to the value of the jid attribute of the service element of type `yahoo` in the active Jabber server's **jabber.xml** configuration file:
			`<service type="yahoo" jid="yahoo.jabber.mycompany.com" name="Y">`
			`<ns>jabber:iq:gateway</ns>`
			`<ns>jabber:iq:register</ns>`
			</service>

ORACLE_SERVICES_ PIM_IM_YAHOO_KEY	The initial group name to assign to users acquired through the Yahoo! transport, if any.	Yahoo	Yahoo Users, Yahoo!
ORACLE_SERVICES_ PIM_IM_MSN	The MSN Instant Messaging transport, if any, configured on the Jabber Server used by the service.	None	msn.jabber.mycompany.com, msn-im.jabber.mycompany.com This name should be set to the value of the jid attribute of the service element of type `msn` in the active Jabber server's **jabber.xml** configuration file <service type="msn" jid="msn.jabber.mycompany.com" name="MSN"> <ns>jabber:iq:gateway</ns> <ns>jabber:iq:register</ns> </service>

Table 18–18 Instant Messaging Module Parameters

Parameter	Description	Default Value	Valid Values
ORACLE_SERVICES_ PIM_IM_MSN_KEY	The initial group name to assign to users acquired through the MSN transport, if any.	MSN	MSN Users, MSN
ORACLE_SERVICES_ PIM_IM_CONFERENCE	The name of the conference service configured on the Jabber Server accessed by the service, if any.	None	conference.jabber.mycompany.com, conf.mycompany.com This name should be set to the value of the jid attribute of the conference element in the active Jabber server's **jabber.xml** configuration file: <conference type="private" jid="conference.jabber.mycompany.com" name="Private Conferencing"/>
ORACLE_SERVICES_ PIM_IM_REFRESH_TIME	The refresh rate for some pages accessed by the service. This value is in milliseconds.	20000	30000 - for 30 seconds 60000 - for 60 seconds

18.1.5.2 Linking to the Instant Messaging Module

You can link to the instant messaging module using the following virtual URL:

omp://oracle/services/pim/instantmessaging

Input Call Parameters of the Instant Messaging Module

The input call parameters of the instant messaging module includes the IMMessage parameter.

Table 18–19 The IMMessage Parameter

Parameter Name	Mandatory	Description	Valid Value
IMMessage	No	The text of a message that is sent through the service.	A string. For example: ■ IMMESSAGE=How are you doing today? ■ IMMESSAGE=I am sending you this message through IM

Output Parameters

An example of the IMMessage output parameter is calling the module to send a simple message. For example:

Input Parameter: IMMessage=Do you want to go see a movie?

18.1.6 Short Messaging

The short messaging module enables users to send messages through such mediums as voice, email, fax or SMS messaging. To send a short message, a user sends the service four parameters: the type of message to be sent (email, SMS, Voice, or Fax), the destination address of the message, the subject text, and the body text of the email. The subject and body text are translated into the medium appropriate to the message type and then sent to the destination.

18.1.6.1 Configuring the Short Messaging Module

This service does not require any third-party software components. It relies on Oracle9*i*AS Wireless transports to be configured. The short messaging modules does not require scripts.

Configuration Parameters

The short messaging service includes the following configuration parameters:

- ORACLE_SERVICES_PIM_SM_FROM_ADDRESS
  - The default address used as sender information for guest or anonymous users.
  - None
  - anonymous@mycompany.com, Company Name, (800)123-4567

18.1.6.2 Linking to the Short Messaging Module

You can link to a short messaging module using the following virtual URL:

```
omp://oracle/services/pim/sm
```

Input Call Parameters

The short messaging module includes the following input call parameters:

Table 18–20 Input Call Parameters of the Short Messaging Module

Parameter Name	Mandatory	Description	Valid Value
type	No	The type of medium through which the message is sent.	The values include: ■ EMAIL (for sending email messages) ■ SMS (for sending a SMS message) ■ VOICE (for sending a message through a phone). ■ FAX (for sending a message through a facsimile)
destinationAddress	No	The address to which the message is sent.	A string. For example: ■ destinationAddress=6505551234 ■ destinationAddress=oraclemobile@oracle.com
subjectText	No	The subject of a message to be sent.	A String. For example: ■ subjectText=Hi There! ■ subjectText=Tomorrow Night
bodyText	No	The body text of a message to be sent.	A String. For example: bodyText=Don't forget to pick up the children on the way home. And buy dinner, too.
sendMessage	No	Specifies whether the service should attempt to send the message with the given information. The service does not send the message unless it has been instructed to do so.	Specify Yes if the service should send the message. Specify No if the service should not send the message.

Output Parameters (Examples)

An example of the short message output parameters is sending a simple message. For example:

Sending an Email

To send an email configure the input parameters as follows:

type=EMAIL

destinationAddress=friend@oracle.com

subjectText=Hey there!

bodyText=How's it going?

sendMessage=yes

Sending a Voice Message

To send a voice message, configure the input parameters as follows:

type=Voice

destinationAddress=9095551234

18.1.7 Document Management

The Oracle Internet File System (iFS) module enables users to both upload files to, or download files from, an Oracle IFS server. The iFs module is integrated with other modules, such as the email module and the fax module.

18.1.7.1 Configuring the iFS Module

The Oracle iFS module enables users to both attach and save files in their native formats. The Oracle iFS module is also integrated with RightFax to enable document printing through faxes. Users can remotely select an attachment and send it by email to another mobile user, who can then view the document (Microsoft Office files) and print it to a nearby fax. This modules does not require scripts.

Required Software

This iFS Module requires the Java development kit for Oracle Internet File System.

Table 18–21 Required Software for the iFS Module

Name	Instructions	Version
Java development kit for Oracle Internet File System	Copy the latest **adk.jar**, **email.jar**, **release.jar**, **repos.jar**, **utils.jar** to $ORACLE_HOME/wireless/lib on Solaris, to %ORACLE_HOME%\wireless\lib on NT. Examples of ORACLE_HOME values: ■ Solaris: ORACLE_HOME=/u01/iaswv20 ■ NT: ORACLE_HOME=d:\oracle\iaswv20	1.1.6 or higher
Oracle Internet File System	Follow the Oracle IFS installation guide.	1.1.6 or higher
Server instance properties file (from the Oracle IFS server instance)	For each IFS server instance, its properties file is needed on the iASW 2.0 server. Copy the properties file (for example: oracle.ifs.server.properties.IfsDefault.properties) so it's included in the OC4J classpath. Example: %ORACLE_HOME/wireless/server/classes	N/A

Configuration Parameters

The iFS module includes the following configuration parameters:

- ORACLE_SERVICES_PIM_IFS_SERVICES
  - Description: A list of IFS service names (they are simply aliases and are up to the user to name), separated by commas.
  - Default Value: ifsserver1, ifsserver2
  - Examples: myifs.oracle.com, myifs2.oracle.com
- ORACLE_SERVICES_PIM_IFS_SERVICENAMES
  - Description: A list of IFS server names, separated by commas. A server name is used when the IFS Java API looks up the client side properties for the IFS server. Therefore, these names must match the names of the properties files.
  - Default Value: Ifs1, Ifs2

  - Examples: IfsDefault1, IfsDefault2 (so the Java API will look for IfsDefault1.properties and IfsDefault2.properties)
- ORACLE_SERVICES_PIM_IFS_SERVICEPASSWORDS
  - Description: A list of IFS schema password (one for each server), separated by commas.
  - Default Value: ifspassword1,ifspassword2
  - Examples: welcome, manager
- ORACLE_SERVICES_PIM_IFS_DOWNLOADDIR
  - Description: The absolute path for the local directory that is used for temporarily storing downloaded IFS files.
  - Default Value: /temp/ifs
  - Examples: d:\temp\ifs

18.1.7.2 Linking to the iFS Module

You can link to the iFS Module using the following virtual URL:

omp://oracle/services/pim/ifs

Input Call Parameters

The iFS module includes the following call parameters and parameter groups:

Table 18–22 The IFSAction Input Parameter

Parameter Name	Mandatory	Description	Valid Value
IFSAction	Yes	The type of action to be performed.	UPLOAD (for uploading a file to the IFS server.) DOWNLOAD (for downloading a filed to the IFS server.) If the value is UPLOAD, then IFSAction requires uploadIfsRequest. If the value is DOWNLOAD, then the downloadIfsRequest output is triggered.

uploadIfsRequest

This optional group includes the following parameters:

Table 18–23 Parameters of the uploadIfsRequest Group

Parameter Name	Mandatory	Description	Valid Value
LOCALPATH	Yes	The absolute local path of the file to be uploaded to the iFS Server.	A string. For example: ■ LOCALPATH=/private/joe docs/file.doc ■ LOCALPATH=c:\TEMP\RESUME.PDF
OBJNAME	No	Enables the user to rename the uploaded file rather than keeping the file name given in LOCALPATH. Note: This name must conform to the UNIX file system convention. For example, it cannot contain the back-slash (\).	A string. For example, OBJNAME=Renamed File.doc

Output Parameters

The Oracle iFS module includes the following output parameters:

downloadIfsInfo

This optional group specifies such information about the downloaded file as the size of the downloaded file, its location, and its original name.

The downloadIFsInfo group includes the following parameters:

Table 18–24 Parameters of the IFsInfo Group

Parameter Name	Mandatory	Description	Valid Value
IFSPATH	Yes	The absolute path of the downloaded file.	A string, for example: ■ IFSPATH=/private/joe/download/file.doc ■ IFSPATH=C:\TEMP\RESUME.PDF

Parameter Name	Mandatory	Description	Valid Value
IFSORIGPATH	Yes	The original IFS path of the downloaded file.	A string, for example: IFSORIGPATH=/ifshome/joe/file.doc
IFSNAME	Yes	The original name of the downloaded filed. This name is provided for display in the user interface.	A string, for example: ■ IFSNAME=file.doc ■ IFSNAME=RESUME.PDF
IFSSIZE	Yes	The size (in kilobytes) of the downloaded file.	Double. For example: IFSSIZE=12.4

Examples

To upload, **files.doc**, from the *directory/private/joe/docs* and save it as **newfile.doc**, you must configure the parameters as follows:

IFSACTION=UPLOAD

LOCALPATH=/private/joe/docs/file.doc

OBJNAME=newfile.doc

To download **files.doc** from the Oracle iFS Server, configure the parameters as follows:

IFSACTION=DOWNLOAD

IFSPATH=/private/joe/download/file.doc

IFSNAME=file.doc

Output Parameter: IFSORIGPATH=ifshome/joe/file.doc

Output Parameter: IFSSIZE=15.0

18.1.8 Fax Module

The fax module enables users to send a fax, check the status of a fax, forward or delete a fax from any wireless device. By combining email or iFS services, it also supports faxing documents through mobile devices.

Requirements

This service requires third-party software components: it uses the RightFax Java API to connect to a RightFax server.

Table 18–25 Required Software for the Fax Module

Name	Instructions	From Version(s)
RightFax Server (available from RightFax)	Install the RightFax server.	7.2
RightFax Integration Module (available from RightFax)	Install the Integration module on fax server.	7.2
RightFax PFD module (available from RightFax)	Install the PFD module on the fax server.	7.2
RightFax Java API (available from RightFax)	Copy RFJava_api.zip(Fax server's RightFax/Production/xml/java directory) to $ORACLE_HOME/wireless/lib on Solaris, to %ORACLE_HOME%\wireless\lib on NT. Include this zip file in the classpath.	7.2

This module does not require scripts.

Sample Cover Page

Since the fax module uses customized cover sheet file, it is highly recommended that you use the provided sample cover page.

To use this cover page, you must have Microsoft Word 2000 installed on your RightFax server for server-side application conversion.

On Solaris installations, this cover page is located at:

```
$ORACLE_
HOME/iaswv20/wireless/j2ee/applications/modules/modules-web
/images/pim/fax/FCS.doc
```

On Windows NT installations, this cover page is located at:

```
%ORACLE_
HOME%\iaswv20\wireless\j2ee\applications\modules\modules-we
b\images\pim\fax\FCS.doc
```

To use the provided fax cover page:

1. Copy the **FCS.doc** to the directory *RightFax\FCS* on the machine in which you installed your RightFax server.
2. Specify which cover sheet to use.
   - Run Enterprise Fax Manager
   - Highlight *Users* under the appropriate server and double-click the user *ID Administrator*. Click the Default Cover Sheets tab. In the Cover Sheet Defaults group box, check Send Cover Sheets and select the cover sheet file (**FCS.doc**) in the Cover Sheet Model field.
   - Highlight Groups under the appropriate server and double-click the group *ID Everyone*. Click the Basic Information tab. Select the cover sheet file (**FCS.doc**) in the Cover Sheet Model field.

Note: See the RightFax Administrator's Guide for detailed instructions on fax cover sheets.

Configuring the Fax Module

The fax module includes the following configuration parameters:

- ORACLE_SERVICES_PIM_FAX_HOST
  - Description: Fax server URL. The Fax module performs fax transactions through this server.
  - Default Value: http://localhost
  - Example: http://144.25.172.183
- ORACLE_SERVICES_PIM_FAX_SENDER
  - Description: The default user to be used when submitting fax transactions to fax server.
  - Default Value: Administrator
  - Valid Values: registered user account in designated fax server
- ORACLE_SERVICES_PIM_FAX_RECORDS_PER_USER
  - Description: The maximum number of fax history records kept in database for each user.
  - Default Value: 20

    - Valid Values: non-negative integers
- ORACLE_SERVICES_PIM_FAX_ITEMS_PER_PAGE
    - Description: The maximum number of fax history records displayed in a page.
    - Default Value: 9
    - Valid Values: non-negative integer
- ORACLE_SERVICES_PIM_FAX_DEBUG
    - Description: Whether to keep log information in the system log file.
    - Default Value: false
    - Valid Values: true, false
- ORACLE_SERVICES_PIM_FAX_LDAP_ENABLED
    - Description: Whether to enable Directory module in searching recipient. Generally set to true only if Directory has fax number information.
    - Default Value: false
    - Valid Values: true, false
- ORACLE_SERVICES_PIM_FAX_DOWNLOADDIR
    - Description: The absolute path for the local directory that is used for temporarily storing downloaded files.
    - Default Value: /temp/fax
    - Valid Values: /private/joe/temp/fax

18.1.8.1 Linking to the Fax Module

You can link to the fax module using the following virtual URL:

omp://oracle/services/pim/fax

The fax module has one input call parameter, FAXTODO. This parameter describes the type of actions to be performed. This mandatory input parameter includes the following values:

Table 18–26 Values of the FAXTODO Input Parameter

Value	Requirement	Triggers Output
NEWFAX	SendNewFax	SendNewFaxResult.
STATUS	faxID	CheckFaxStatusResult
DELETE	faxID	deleteFaxResult
FWD	forwardFax	forwardFaxResult

sendNewFax

The FAXTODO parameter includes sendNewFax group. This mandatory group of parameters specify the information about the fax to be sent.

Table 18–27 Parameters of the sendNewFax Group

Parameter Name	Mandatory	Description	Valid Value
SENDER_NAME	No	Sender name.	A string. For example: SENDER_NAME=Joe Smith
SENDER_CORP	No	Sender company.	A string. For example: SENDER_CORP=Oracle Corp.
SENDER_PHONE	No	Sender phone number.	A string. For example: SENDER_ PHONE=1(650)123-4567
SENDER_FAX	No	Sender fax number.	A string. For example: SENDER_ FAX=1(650)123-4567
SENDER_ADDRESS	No	Sender address.	A string. For example: SENDER_ADDRESS=Home address
SENDER_NOTES	No	Other sender information not listed above.	A string. For example: SENDER_NOTES=email: joe.smith@oracle.com
RECIPIENT_NAME	No	Recipient name.	A string. For example: RECIPIENT_NAME=John White
RECIPIENT_CORP	Yes	Recipient company.	A string. For example: RECIPIENT_ CORP=1(650)123-4567

Parameter Name	Mandatory	Description	Valid Value
RECIPIENT_PHONE	No	Recipient phone number.	A string. For example: RECIPIENT_ PHONE=1(650)987-6543
RECIPIENT_FAX	Yes	Recipient fax number.	A string. For example: RECIPIENT_ FAX=1(650)123-4567
RECIPIENT_ADDRESS	No	Recipient address.	A string. For example: RECIPIENT_ ADDRESS=Work address
MESSAGE	No	Short message to be written on cover page.	A string. For example: MESSAGE=An awesome resume!
ATTACHMENT	No	Attachment to be faxed.	A string. For example: ATTACHMENT=mydoc/resume.pdf

forwardFax

The FAXTODO parameter includes the forwardFax group. This mandatory group includes the following parameters:

Table 18–28 Parameters of the forwardFax Group

Parameter Name	Mandatory	Description	Valid Value
FAXID	Yes	The unique id of the fax to be forwarded.	A string. For example: FAXID=12345
RECIPIENT_FAX	Yes	The destination fax number.	A string. For example: Example: RECIPIENT_ FAX=1(650)123-4576

Output Parameters

The Fax module includes the following output parameters:

Table 18–29 Output Parameters of the Fax Module

Parameter	Mandatory	Description	Valid Value
sendNewFaxResult	Yes	Whether the fax was successfully sent or not.	A string. For example: sendNewFaxResult=Fax has been successfully submitted for sending.
checkFaxStatusResult	Yes	The fax status.	A string. For example: checkFaxStatusResult=Sending (50%)
deleteFaxResult	Yes	Whether the fax was successfully deleted or not.	A string. For example: deleteFaxResult=Fax successfully deleted.
forwardFaxResult	Yes	Whether the fax was successfully forwarded or not.	A string. For example: forwardFaxResult=Fax has been successfully submitted for forwarding.

Examples

To send a fax, you configure the FAXTODO parameters as follows:

FAXTODO = NEWFAX

RECIPIENT_FAX = 1(650)987-6543

MESSAGE = Hello world!

sendNewFaxResult = Fax has been successfully submitted for sending.

To check the status of a fax you configure the FAXTODO parameters as follows:

FAXTODO = STATUS

faxID = 16543

checkFaxStatusResult = OK

18.1.9 Tasks

The tasks module enables users and applications to schedule and manage tasks. This module integrates with Lotus and Exchange servers.

Requirements

This service implements two distinct modes, both with the same user experience but with different back-ends. In its Lotus Domino mode, it fully integrates with a Lotus Domino server to enable mobile Domino users. In its Microsoft Exchange Mode, it fully integrates with a Microsoft Exchange server to mobile enable Exchange users.

The Tasks module requires the following third-party software:

Table 18–30 Software Required for the Tasks Module

Name	Instructions	From Version
MS Exchange	[Exchange Mode] Install the Microsoft Exchange Server.	5.5
MS IIS	{Exchange Mode] Install the Microsoft Internet Information Server.	4.0
MS CDO	{Exchange Mode] Collaborative Data Objects. Available with Exchange SDK. The cdo.dll library must be installed on the IIS Server.	1.2.1
Oracle9*i*AS Wireless 2.0 Exchange ASP(Shipped with Oracle9*i*AS Wireless)	[Exchange Mode] Create any directory on the IIS server. Copy all the files in $ORACLE_HOME/iaswv20/wireless/j2ee/applications/modules/modules-web/pim/tasks/asp/ for a Solaris install, %ORACLE_HOME%\iaswv20\wireless\j2ee\applications\modules\modules-web\pim\tasks\asp\ for an NT install to just created directory on the IIS server. Invoke the properties dialog box for this directory. Choose the Directory tab. Click the "Edit" button for the "Anonymous Access and Authentication Control", make sure the following check boxes are set as follows: ■ Allow Anonymous Access - unchecked ■ Basic Authentication - checked ■ Windows NT Challenge/Response - checked	

Name	Instructions	From Version
Lotus Java SDK	[Domino Mode] Install the Lotus Domino Toolkit for Java / CORBA, add NCSO.jar to the classpath for Oracle9*i*AS Wireless 2.0. Installing the toolkit creates a "DTJava" directory on the file systemCopy the DTJava / lib / NCSO.jar to $ORACLE_ HOME / wireless / lib on Solaris, to %ORACLE_HOME% \ wireless \ lib on NT. Examples of ORAClLE_HOME values: ■ Solaris: ORACLE_ HOME= / u01 / iaswv20 ■ Windows NT: ORACLE_ HOME=d: \ oracle \ iaswv20 Ensure you download "Lotus Domino Toolkit for Java / CORBA Release 5.0.8 Update" or "Lotus Domino Toolkit for Java / CORBA Release 5.0.5 Update Shipping". Do not use version 2.x toolkit. On the Domino server, the server tasks HTTP and DIIOP must be running. Ensure that the Domino server notes.ini file contains the following line: `ServerTasks=<other tasks>,http,diiop`	R5

This module does not require scripts.

Configuring the Task Module

The Task Module includes the following configuration parameters:

- ORACLE_SERVICES_PIM_TASKS_CLASS
  - Description: The service implementation class, which determines whether the Domino or Exchange mode is used.
  - Default Value: None
  - Valid Values: oracle.panama.module.pim.tasks.exchange.ExchangeTaskService for Exchange server

oracle.panama.module.pim.tasks.domino.DominoTasksService for Lotus Domino server.

- ORACLE_SERVICES_PIM_TASKS_SERVER_NAME
  - Description: Designates the Exchange server or Domino server host name.
  - Default Value: None
  - Valid Values: exchange.mycompany.com
    domino-server.mycompany.com
- ORACLE_SERVICES_PIM_TASKS_SERVER_DATA_WEB_LINK
  - Description: If Exchange mode is selected, this is the URL pointing to the Oracle9*i*AS Wireless Exchange Tasks ASP (Tasks.asp) running on IIS.
  - Default Value: None
  - Valid Values:
    http://iis.mycompany.com/Oracle9iASW20/exchange/Tasks.asp
- ORACLE_SERVICES_PIM_TASKS_DOMAIN
  - Description: The domain attached to the user account information. This is free form field. If the Oracle9*i*AS Wireless instance supports multiple tasks servers, then these must have the same domain. Different PIM services with the same domain share the user account information.
  - Default Value: None
  - Valid Values: DominoDomain for the Domino Mode
    ExchangeDomain for Exchange.

18.1.9.1 Linking to the Task Module

You can link to the task module using the following virtual URL:

omp://oracle/services/pim/tasks

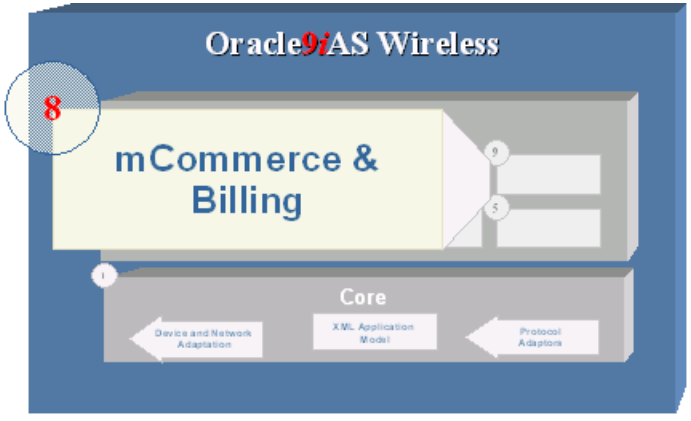
19

m-Commerce

This document describes reusable Services that are included in Oracle9*i*AS Wireless. Each section of this document presents a different topic. These sections include:

- Section 19.1, "m-Commerce Service"
- Section 19.2, "m-Commerce APIs"
- Section 19.3, "Mobile Wallet (m-Wallet)"
- Section 19.4, "Translator"
- Section 19.5, "iPayment"
- Section 19.6, "Formfiller"
- Section 19.7, "Creating a Billing Mechanism"

Figure 19–1 M-Commerce



19.1 m-Commerce Service

Oracle m-Commerce Service is a set of Oracle9*i*AS Wireless modules that securely store user profiles, supply information authorized by users of third party applications, and interface with on-line payment mechanisms to complete transactions. The m-Commerce Service also translates existing WML applications into Mobile-XML, and uses FormFiller to map forms, which spares users from entering information from a mobile device. Oracle m-Commerce Service is automatically installed along with Oracle9*i* Application Server.

The extendible modules architecture on the m-Commerce Service enables the development of drivers to integrate m-Commerce services to third-party applications.

19.2 m-Commerce APIs

You can build an m-Commerce application using Oracle9*i* Application Server Mobile XML. You can incorporate any m-Commerce component to this application by adding URL links to the modules complying with their APIs.

If you have already developed an m-Commerce application in WML, you can run it through the Translator Module by calling its API, and providing your application's URL. This action will add links from your application to all m-Commerce modules.

19.2.1 Before You Begin

Before you configure the modules, you must do the following:

1. Modify the **security.sh** script and change the variables. See Section 19.3.1.5 for a list of variables.
2. Set the JAVA13_HOME environment to the current JDK directory.
3. Run the script to generate the key, **security.sh** (UNIX) or **security.bat** (WINDOWS).
4. Record the path of the encryption file %ORACLE_HOME/wireless/j2ee/applications/modules/modules-web/commerce/setup/scripts KeyMgmtProps.enc
5. Using the Edit Master Services function of the Service Designer, modify the default value of the input parameter, ORACLE_SERVICES_COMMERCE_SECURITY_PROPS_PATH.

Note: The default value of this input parameter must not include the **KeyMgmtProps.enc** file.

19.3 Mobile Wallet (m-Wallet)

The mobile Wallet (m-wallet) enables users to manage their profile from mobile devices as well as participate in commerce transactions and track their activity.

The m-Wallet Module securely stores user's payment instrument information, such as credit cards, bank accounts, and shipping addresses. Upon user approval, other m-Commerce applications can retrieve this information to process payments.

The Oracle9*i*AS Wireless administrator can configure the Credit Cards, Bank Accounts and Extended Information compartments at any time, even if they contain values that users have previously entered. The fixed compartments are profile, shipping addresses and internet accounts.

The m-Wallet is divided into compartments that can hold one or more instruments. For example the Credit Cards compartment holds as many credit cards as a user sees fit to enter. The Extended Information compartment, however, holds only one information set.

19.3.1 Configuring the m-Wallet

Mobile Wallet module (m-Wallet) provides a convenient single-click commerce payment mechanism. It is a server side, encrypted entity that contains payment instrument, identification and address information for registered users. m-Wallet enables users to store all the information required to fill out commerce-related forms from any application. That information is used to complete transactions, and through APIs (built and maintained by authorized third-party service providers), can be made available to authorized partners and e-merchants. It processes requests (via proxies) for personal and payment instrument information issued through HTML or WML forms by third-parties, and presents them to users, who decide explicitly what information gets sent back to the third-party. The wallet stores this information securely for users, providing them an easy, secure shopping experience, and freeing them from repeatedly entering information.

m-Wallet also encrypts and decrypts all of the information stored in the Repository using a three-part key comprised from a combination of the following:

- a system key (specific to each deployment of the product)

- a user-specific key (uniquely identifying users within the system, and retrieved when a function is applied to specific user information)
- the user's trading password

Each portion of the three-layer key can be changed independently, but each of them is required in order to decrypt wallet-stored information. This combination is never stored, only an encrypted alias, assigned to each entry during its creation or modification, is sent over the wireless network.

Because security is central to the m-Wallet you must configure HTTPs for the m-Wallet.

19.3.1.1 Configuring the OC4J Application Server for HTTPS

The installation of Oracle9*i* Application Server includes both installation of a dummy certificate and automatic HTTPS configuration. Use this certificate only for testing and development, as it is not signed by a trusted entity.

Note: When setting up a production machine, you must install the actual certificate for HTTPS on the server.

19.3.1.2 Configuring the SQL Tables

You do not need to configure the SQL tables; installing Oracle9*i*AS Wireless installs all of the tables needed by the Formfiller module.

19.3.1.3 Configuring the Security Server

You must install and configure the Secure Key Server before using the Wallet Module. For more information, see Section 19.2.1.

19.3.1.4 Java Configuration

The following security **.jar** files must be placed under jdk1.3/jre/lib/ext. (The Oracle Installer creates this directory.)

- ojcae.jar
- US_export_policy.jar
- local_policy.jar
- jce1_2_1.jar

19.3.1.5 Scripts for Generating and Installing the Security Keys

You must insert an encryption key. This encryption key forms the backbone for encryptions and decryptions performed by the system.

To configure security.sh, complete these steps:

1. Modify the security.sh script and change the variables.
2. Run the script to generate the key, security.sh (UNIX) or security.bat (WINDOWS).

To generate and install the security key, follow these steps:

1. Go to ORACLE_HOME / wireless / sample
2. Edit the template **security.sh** (or **security.bat**) script to configure the following:

Table 1 Security Key configuration

DB_USER, DB_PWD and DB_URL	Passwords for the Wireless database schema are randomized out of the box, and are not available to end users. Hence, the password must be changed by a user through the Oracle Enterprise Manager console; this new password must be used for security key configuration. The password for the Wireless schema can be changed from the Oracle Enterprise Manager console: 1. Click on the link corresponding to the middle tier. 2. Click the 'Configure Schema' option. 3. Select the radio button corresponding to 'Oracle9*i*AS Wireless'. 4. Click 'Change Password'. All platforms are affected by this.
SEC_SEED	The seed number used to create the System Encryption Key. The range of the number is that of a java long number (that is, -922337203685477580**8**L to 922337203685477580**7**L.) For example: SEC_SEED 1234567890
SEC_PWD	The password with which the generated System Encryption Key is encrypted and stored in the database. The password can be any length greater than eight characters. For example: SEC_PWD systemEncryptionKeyPassword
SEC_FILE_PWD	The password with which to encrypt the local **KeyMgmtProps.en**c file (which contains secure information) protecting it from unauthorized access. The password can be any length greater than eight characters. For example: SEC_FILE_PWD fileEncryptionPassword.

3. Save your changes and execute the script. This generates a secure file, **KeyMgmtProps.enc** in the current directory and also generates, encrypts and inserts the System Encryption Key into the database as well as printing out the directory path for the security file. You must save the path to the file because it is used as a service input parameter value.

   Note: Because the script contains sensitive information, you should destroy it after running it or move it to a secure place.

19.3.1.6 Configuring modules.properties

You must configure the **modules.properties** file with the correct paths for both the HTTP and HTTPS listeners so that such resources as audio and images are presented properly in HTTPS mode.

The **modules.properties** is located under ORACLE_HOME/wireless/server/classes/messages/oracle/panama/module/common

An example of the correct values for **modules.properties** is as follows:

```
/*
 *
 * $Copyright:
 * Copyright (c) 2001 Oracle Corporation all rights reserved
 */
#--------------------------------------------------------------------------
# Please configure here the module server URLs.
# If no value if configured, we'll try to use the request in order to
# get the server name
# example:
# device.resources.host=http://myserver.com
# device.resources.port=9080
# device.resources.secure.host=https://myserver.com
# device.resources.secure.port=9081
#--------------------------------------------------------------------------
device.resources.host=http://www.myserver.com
device.resources.port=9080
device.resources.secure.host=https://www.myserver.com
device.resources.secure.port=443
```

19.3.1.7 Service Input Parameters

There are two optional service input parameters which do not require configuration.

ORACLE_SERVICES_COMMERCE_WALLET_CONFIRMATION_PAGE

Whenever a third party application requests user information from the Wallet, the user must agree to share this information. This parameter is set regardless of whether this confirmation card is presented to user.

The valid values for this input parameter include:

- ALWAYS: Always show the confirmation card user. The user cannot override this value.
- NEVER: Never show the confirmation card to the user and automatically return the user information to the third party application.
- USER: Unless otherwise specified by the user, always show the confirmation card to the user. This is the default value.

Administrators may want to customize these values depending on the site's policy.

ORACLE_SERVICES_COMMERCE_WALLET_IS_SECURE

Defines whether the m-Wallet module runs in HTTP or HTTPS.

The valid values for this input parameter include the following:

- true: When accessing m-Wallet module, the connection between the user device and Oracle9*i*AS Wireless will be secure HTTPS. This is default value.
- false: When accessing wallet module, the connection between the user device and Oracle9*i*AS Wireless will be non-secure HTTP.

Wallet Module requires the Security Server to be correctly installed and configured prior to its use. There is one required parameter which requires configuration:

ORACLE_SERVICES_COMMERCE_SECURITY_PROPS_PATH

- Description: The fully qualified path to the directory where the **KeyMgmtProps.enc** (encrypted file) was created, not including the **KeyMgmtProps.enc** name itself.
- Valid Value: For this service input parameter is: /private/ptg20/iaswv20/wireless/j2ee/applications/modules/modules-web/commerce/setup/scripts

- Default Value: /
  Note: The user must customize this value before using Wallet Module. For more information see Section 19.2.1.

19.3.2 Linking to the M-Wallet

You can link to the m-wallet using the following virtual URL:

omp:/ / oracle / services / commerce / wallet

The m-wallet includes the following input call parameters:

Wallet_Action

Wallet_Action is used to determine the type of overall action that service requests. This is a mandatory parameter.

Table 19–2 Input Parameters for Wallet_Action

Valid Value	Description	Requirement
GETSTRUCTURE	Used to retrieve the Wallet structure definition. Triggers WALLET_STRUCTURE	.
GET_FORM_DATA	Used when a third party application wants to request information from the user's mobile wallet.Triggers output generateUserResponse	`getWalletInfoRequest`.
GET_INET_ACCT	Used to add Internet account information in the user's wallet.	`createInternetAccountRequest`.
GEN_USER_PASS	Used to automatically generate the username and password information. Triggers output generateUserResponse.	

getWalletInfoRequest

This group contains the following parameters. This is an optional group.

Table 19–3 Parameters of the getWalletInfoRequest Group

Parameter Name	Mandatory	Description	Valid Value
FORM_TITLE	Yes	This parameter is displayed as part of the Wallet module for the duration of the call.	A string. For example: FORM_TITLE=Movie Ticket Purchase.

Parameter Name	Mandatory	Description	Valid Value
GET_DATA	Yes	A comma-separated string of tokens which specify which values to retrieve from the wallet.	Valid values in this string are: ■ CC (triggers output creditCardData) ■ BA (triggers output bankAccountData) ■ FN (triggers output FIRSTNAME) ■ LN (triggers output LASTNAME) ■ EMAIL (triggers output EMAIL) ■ PHONE (triggers output phoneData) ■ INT_ACC (triggers output internetAccountData) ■ SHIP (triggers output shippingData) For example: ■ GET_DATA=FN,LN,SHIP ■ GET_DATA=CC, PHONE, INT_ACC
APPICATION	No	The application name displayed to the user and stored in the History file, so that the user always knows which applications are requesting the user's wallet information.	A string. For example: APPLICATION=Bookshop Application.
ISEXCLUSIVE	No	If set to True, then the user can chose either Credit Card or Bank Account. This parameter is used only by the Payment module.	A boolean.
DOMAIN	Yes		A string. For example: DOMAIN=http:/wwww.oraclemobile.com
ACCOUNT_ ID	Yes		A string. For example: ACCOUNT_ ID=smurgle.
PASSWORD	Yes		A string. For example: PASSWORD=237894.

19.3.3 Output Parameters for the m-Wallet

The m-Wallet module includes the following output parameters:

Table 19–4 Output Parameters for the m-Wallet Module

Parameter Name	Mandatory	Description	Valid Value
WALLET_SELECT	Yes	Defines whether the operation completed correctly. If the user cancels the wallet operation, this variable contains False.	Valid values are True and False
WALLET_STRUCTURE	No	This string specifies the wallet's internal structure. The wallet structure is based on fixed- and user- defined compartments. The fixed compartments include the User Profile, Internet Accounts, and Shipping Addresses. The user-defined compartments include Credit Card, Bank Account, and Extended Info. Restriction: The return string is formatted as COMPARTMENT_NAME:FIELD_ NAME90FIELD_DESCRIPTION.	A string. For example: WALLET_STRUCTURE=If the wallet has compartments CC and BA for credit card and bank account respectively, then the return string can be CC:CCNUM()CreditCard Number, CC:CCEXP()Credit Card Expiration Date, BA:BNUM() Bank Account Number...
FIRSTNAME	No	This variable holds its value of the user's first name when the calling application requests the user's name. This variable cannot be changed, as it is part of the fixed Profile compartment.	A string. For example: FIRSTNAME=John
LASTNAME	No	This variable holds the value of the user's last name when the calling application requests the user's last name. It cannot be changed as it is part of the fixed Profile compartment.	A string. For example: LASTNAME=John
EMAIL	No	This variable holds the value of the user's email address when the calling application requests the user's email. This cannot be changed, as it is part of the fixed Profile compartment.	A string. For example: EMAIL=John.Doe@company.com

CreditCardData

The Credit Cards structure held in **wallet.properties**. The fields are returned as request parameters. The following parameters are the default parameters of the CreditCardData group. This is an optional parameter.

Table 19–5 Parameters of the CreditCardData Group

Parameter Name	Mandatory	Description	Valid Value
CC	Yes	A short name for the credit card.	A string. For example: CC=My Bank Visa Card.
CC_HOLDER_NAME	Yes	The name of the holder of the credit card.	A string. For example: CC_HOLDER_NAME=John Doe
CC_HOLDER_ADDRESS_LANDMARK	Yes	The billing address of the holder of the credit card. This is a link to the user's locationmarks. Restriction: this landmark must be defined in the location module.	A string. For Example: CC_HOLDER_ADDRESS_LANDMARK=Office at Oracle
CC_EXPIRATION_DATE	Yes	The expiration date of the credit card. Restriction: this should be in the MM/YYYY form. This also must be defined in **wallet.properties**.	A string. For example: CC_EXPIRATION_DATE=04/2003
CC_LANDMARK_NAME	Yes	The locationmark of the credit card. Restriction: the parameters for the street address (such as CC_ADDRESS_LINE1)are built on-the-fly as Wallet Module 'knows' that Billing Address is a reference to a location mark.	A string. For example: CC_LANDMARK_NAME=Office at Oracle
CC_ADDRESS_LINE1	No		A string. For example: CC_ADDRESS_LINE1=500 Oracle Pkwy

Parameter Name	Mandatory	Description	Valid Value
CC_ADDRESS_LINE2	No		A string. For example: CC_ ADDRESS_LINE2=
CC_CITY	No		A string. For example: CC_ CITY=Redwood Shores
CC_STATE	No		A string. For example:CC_ STATE=CA
CC_COUNTRY	No		A string. For example: CC_ COUNTRY=USA
CC_ZIPCODE	No		A string. For example: CC_ ZIPCODE=94065

bankAccountData

The Bank Account structure defined in **wallet.properties.** All the fields are returned as request parameters.

This group contains the following parameters. This is an optional group.

Table 19–6 Parameters of the bankAccountData Group

Parameter Name	Mandatory	Description	Valid Value
BA	Yes	The short name for the bank account	A string. For example: BA=Checking ****-2438
BA_HOLDER_NAME	Yes	The name of the holder of the bank account.	A string. For example: BA_ HOLDER_NAME=John Doe
BA_HOLDER_ADDRESS_LANDMARK	Yes	Statement Address - this is a link to the user's Location Marks Restriction: This landmark must be defined in the location module.	A string. For example: BA_ HOLDER_ADDRESS_ LANDMARK=Palo Alto branch of Western Union
BA_ACCT_NUMBER	Yes	The number of the bank account. Restriction: this can only be numbers; all other characters are ignored.	A string. For example: BA_ ACCT_NUMBER=23894592

Parameter Name	Mandatory	Description	Valid Value
BA_ACCT_TYPE	Yes	The type of account, such as checking or savings.	Checking, Savings,Market-Rate. For example: BA_ACCT_ TYPE=Checking
BA_FI_ROUTING_NUMBER	Yes	The routing number of the bank. Restriction: this must only be numbers; all other characters are ignored.	A string. For example: BA_FI_ ROUTING_ NUMBER=23985002394
BA_FI_NAME	Yes	The name of the bank.	A string. For example: BA_FI_ NAME=Bank of America
BA_LANDMARK_NAME	Yes	The parameters for the bank's street address (such as BA_ ADDRESS_LINE1) are built on-the-fly, as the Wallet module 'knows' that Billing Address is a reference to a location mark. Restriction: This landmark must be defined in the location module.	A string. For example: BA_ LANDMARK_NAME=Palo Alto branch of Western Union
BA_ADDRESS_LINE1	No		A string. For example: BA_ ADDRESS_LINE1=2035 Island Parkway
BA_ADDRESS_LINE2	No		A string. For example: BA_ ADDRESS_LINE2=Apt. #P-24
BA_CITY	No		A string. For example: BA_ CITY=Menlo Park
BA_STATE	No		A string. For example: BA_ STATE=CA
BA_COUNTRY	No		A string. For example: BA_ COUNTRY=USA
BA_ZIPCODE	No		A string. For example: BA_ ZIPCODE=91750

idData

The Extended Information structure defined in **wallet.properties.** All fields are returned as request parameters.

The idData group contains the following parameters. This is an optional group:

Table 19–7 Parameters of the idData Group

Parameter Name	Mandatory	Description	Valid Value
ID_SSN	No	the Social Security Number	A string. For example: ID_SSN=298459825
ID_DL	No	A driver's licence number	A string. For example: ID_DL=B239922023
ID_DL_STATE	No	The state in which the driver's license has been issued.	A string. For example: ID_DL_STATE=CA
ID_DL_EXP_DATE	No	The expiration date of the driver's license. Restriction: The format (*MM/DD/YYYY,*) is defined in the **wallet.properties**.	A string. For example: ID_DL_EXP_ DATE=04/27/2007
ID_PASSPORT	No	A passport number	A string. For example: ID_ PASSPORT=B293A923CK
ID_PASSPORT_ EXP_DATE	No	The expiration date of the passport. Restriction: The format *(MM/DD/YYYY)* is defined in the **wallet.properties**.	A string. For example: ID_PASSPORT_EXP_ DATE=04/08/1997

19.3.3.1 Extending the m-Wallet Structure

You can configure the structure of the m-Wallet so that its contents can be personalized according to usage.

The m-Wallet structure is defined in the **wallet.properties** file located under the directory ORACLE_
HOME/wireless/server/classes/messages/oracle/panama/module/commerce/wallet/wallet.properties

This file contains the definitions for credit cards, bank accounts and extended information. In addition, this file contains the definition of the formats to be used for each field. The format definitions are used for internationalization purposes of the dates.

Defining a Compartment

To define a compartment, When defining a compartment, there are few things one needs to do:

1. Add a reference to this compartment in the compartments key:

```
compartments=CREDIT_CARD,BANK_ACCOUNT,ID
```

2. Add the total number of fields in this new compartment:

```
CREDIT_CARD.fieldnumber=6
```

3. Add all the fields for this compartment and add attributes for each field. You can add up to six attributes (0 - 5)

   The variable is built as follows:

   ```
   <comparment_name>.fieldNN.itemNN=<value>, where:
   compartment_name = current compartment name, i.e. CREDIT_CARD
   fieldNN = represents the current field, starting in 1, i.e. CREDIT_
   CARD.field1
   itemNN = represents each attribute of this field, starting in 0, i.e.
   CREDIT_CARD.field1.item0
   ```

   The attributes are defined as follows:

   – The application reads variables from the request to retrieve a value for an specific field from the wallet. This variable name is defined in the attribute #0

     `CREDIT_CARD.field1.item0=<request_variable_name, i.e. CC_HOLDER_NAME>`

   – The label that appears to the end user is defined in the attribute #1. It is a key to a value defined in **portal.properties** (for internationalization purposes).

     `CREDIT_CARD.field1.item1=<key.in.portal.properties, i.e. modules.commerce.wallet.creditcard.holdername`

– Each field can be either optional or mandatory, depending on the compartment rules. This is defined in attribute #2.

```
CREDIT_CARD.field1.item2=<MANDATORY|OPTIONAL>
```

– The format of this field (for display on WML and HDML WAP devices) is defined in attribute #3 and is a reference of a format previously defined in **wallet.properties**

```
CREDIT_CARD.field1.item3=<format, i.e. MIXED_FORMAT,
NUMBER_FORMAT, DATE_FORMAT>
```

– If the field contains a list of possible values, such as credit card types, then they are listed in attribute #4. Use a comma (,) to separate these values.

```
CREDIT_CARD.field1.item4=<comma-separated list of
values, i.e. Visa, Master, AmEx, Discover, Diners>
```

– Attribute #5 is used if the current field stores an address by having a reference to an existing location mark.

```
CREDIT_CARD.field1.item5=<LINK_LOC>
```

19.4 Translator

The Translator module enables any site written in WML to be rendered on any device by converting its contents to MobileXML. It also enhances the navigation of sites originally authored in WML by adding links to Oracle9*i*AS Wireless core services. Currently, only WAP sites are supported. There is no output parameter; the translated result and status code are internally consumed by the translator module.

19.4.1 Configuring the Translator Module

The Translator has the following service input parameters:

- ORACLE_SERVICES_COMMERCE_TRANSLATOR_DEFAULT_CONNECTION
  - Description: A fully qualified class name. The class will be instantiated automatically and the instantiated object is used to get content from an URL. This default class uses HTTP connection.
  - Default Value: oracle.panama.module.commerce.translator.WMLConnectionImpl
  - Customizability: No configuration is necessary.

- ORACLE_SERVICES_COMMERCE_TRANSLATOR_HELPER_WML
  - Description: A fully qualified class name. The class will be instantiated automatically and the instantiated object is used to transform the WML document into MobileXML document.
  - Default Value: oracle.panama.module.commerce.translator.WMLTransformImpl
  - Customizability: No configuration is necessary.
- ORACLE_SERVICES_COMMERCE_TRANSLATOR_XSL_WML_FILENAME
  - Description: AA URL pointing to the location of the XSL stylesheet used to transform WML into MobileXML.
  - Default Value: Null--to use the default location for the XSL http://server.com/xsl/wml_to_mxml.xsl
  - Customizability: Users may change this value if they want to provide a different URL for the XSL file.

19.4.2 Linking to the Translator Module

You can link to the Translator module using the following virtual URL:

omp://oracle/services/commerce/translator

The Translator module includes the following input call parameters.

Table 19–8 Input Call Parameters of the Translator Module

Parameter Name	Mandatory	Description	Restriction	Valid Value	Example
XLTORSITE	Yes	The source URL of the WML site, whose content will be translated into MobileXML	This must be in a valid URL format.		XLTORSITE =http://www.oracle mobile.com
XLTORLANG	No	The source language of the WML site	Valid strings are WML, HDML, VXML. Currently only WML is accepted.	WML	

Parameter Name	Mandatory	Description	Restriction	Valid Value	Example
EXTENSIONACTION	No	Extension actions are actions other than translating a URL. The extension action can be either "HELP" (used to show help message) or "DELPRESET" (used to delete a preset). If the extension action is passed as "DELPRESET", then the preset label should be passed.		The valid values include: HELP (Help message for translator service) DELPRESET (Delete a preset)	
PRESETLABEL	No	Label of the preset that will be deleted, which is normally a site name	The label should be a string.		PRESETLABEL=www.oraclemobile.com

Output Parameters

This section includes invocation examples for translating a site and removing a preset using the following input parameters:

- XLTORSITE
- EXTENSIONACTION
- PRESETLABEL

Translating a Site

You use the input parameter XLTORSITE to translate a WML site as follows:

XLTORSITE=http://www.oraclemobile.com

Removing a Preset

You use the input parameters EXTENSIONACTION and PRESETLABEL to remove a preset as follows:

EXTENSIONACTION=DELPRESET

PRESETLABEL=www.oraclemobile.com

19.5 iPayment

The iPayment module, which integrates with Oracle CRM iPayment module, processes credit card and bank account transactions.

Payment Processing enables integration with payment mechanisms, such as Oracle's CRM iPayment. As a result, credit card processing and bank account transactions are carried out through direct connections to financial networks. You can add other drivers that integrate payment solution providers per customer requests.

Through integration with Oracle CRM's iPayment component, which implements transaction settlement support for credit cards and bank accounts, allows transactions to be processed directly through the platform rather than through a processing infrastructure deployed by merchants.

19.5.1 Configuring the iPayment Service Module

You must correctly install and configure the Oracle CRM iPayment before you use the iPayment module.

19.5.1.1 Service Configuration Parameters

The iPayment Service module includes the following service configuration parameters:

Note: ORACLE_SERVICES_COMMERCE_PAYMENT_DBCFILE and ORACLE_SERVICES_COMMERCE_PAYMENT_ECAPPID are required parameters.

- ORACLE_SERVICES_COMMERCE_PAYMENT_DBCFILE
  - Description: This value points to the location of the DBC file, used by Oracle CRM iPayment. This file has the necessary configuration for the iPayment database, such as username and password.
  - Valid Values: /apps.dbc (if DBC file is in root directory); d:/iaswv20/wireless/j2ee/applications/modules/modules-web/payment/apps.dbc
  - Default Value: /apps.dbc

    - Customizability: Users must customize this value before using the iPayment module.
- ORACLE_SERVICES_COMMERCE_PAYMENT_ECAPPID
    - Description: This value represents the Electronic Commerce Application ID (ECAPPID) within iPayment. An ECAPPId is the Id by which iPayment identifies the calling application. All applications in 11i have a unique "Application Id" by which it is identified. Users of payment module will need to register a new ECAPPID for Oracle9*i*AS Wireless.
    - Default Value: 00000
    - Customizability: The user must customize this value before using the iPayment module.
- ORACLE_SERVICES_COMMERCE_PAYMENT_DEFAULT_TRANSACTION
    - Description: The default transaction processor which is used for such functions as creating accounts, submitting transaction requests, cancelling transactions, and querying transactions. The default class (`OracleIPaymentHook`) provides the driver for Oracle CRM 11i iPayment.
    - Default Value: oracle.panama.module.commerce.payment.OracleIPaymentHook
    - Customizability: Only needs to be changed if you're providing a different payment system.
- ORACLE_SERVICES_COMMERCE_PAYMENT_DEFAULT_CURRENCY
    - Description: The default currency that is used when submitting transactions to Payment Module. **Note**: the currency may also be configured at runtime by sending it in the request. It is used for that transaction only.
    - Valid Values: USD - for US dollar; BRL - for Brazilian Real
    - Default Value: USD
    - Customizability: Users can customize this to their default currency.

19.5.1.2 Capturing Transactions

Merchants can use a URL whenever they want to capture previously authorized transactions. This URL can be used in both secure and non-secure modes. The difference between the two modes is the HTTP and HTTPS protocols.

Non-Secure Capture

The http URL for the non-secure capture of a previously authorized transaction is as follows:

http://myserver.com:9080/modules/commerce/payment/jsp/IPaymentProcess.jsp?

MERCHANTID=<merchantID>&

MERCHANTPW=<merchantPWD>&

TRXID=<transactionID>&

CURRENCY=<currency>&

AMOUNT=<amount>

For a merchant called *BookStore* to capture transaction #1234 in the amount of US$100.00, you call the URL and then enter the parameters as follows:

http://myserver.com:9080/modules/commerce/payment/jsp/IPaymentProcess.jsp?

MERCHANTID=bookstore&MERCHANTPW=welcome&TRXID=1234&CURRENCY=USD&AMOUNT=100

Secure Capture

In order to use the secure mode for the capture URL, you must first configure the SSL for the OC4J Application Server. For information on configuring the OC4J Application Server, see Section 19.3.1.1.

The HTTPS URL for the secure capture of a previously authorized transaction is as follows:

https://myserver.com:443/modules/commerce/payment/jsp/IPaymentProcess.jsp?

```
MERCHANTID=<merchantID>&
MERCHANTPW=<merchantPWD>&
TRXID=<transactionID>&
CURRENCY=<currency>&
AMOUNT=<amount>
```

Note: Merchants must have an Oracle9*i*AS Wireless account to use the capture URL.

19.6 Formfiller

The Formfiller module is a self-teaching form filler, one that maintains mappings between application form fields and wallet elements. The Formfiller accepts a URL and a list of label and variable names as input parameters, and checks if there is a stored mapping from the given labels and variables to wallet fields. If there is no such mapping, it enables users to create a new mapping into wallet fields. Once a mapping is retrieved or created, it calls the wallet, asking it for the given mapped information. Upon successful completion, the module returns a status of Success along with the wallet values corresponding to the label / variable name list. Otherwise, a status code of Failure will be returned

19.6.1 Configuring the Formfiller Module

Before you can deploy the Formfiller module, you must install the Formfiller, configure the guessing heuristics, and approve the mappings.

19.6.1.1 Installing Formfiller

You do not need to configure the SQL tables; installing Oracle9*i*AS Wireless installs all of the tables needed by the Formfiller module.

19.6.1.2 Configuring the Guessing Heuristics

When an existing mapping is not available, the Formfiller enables authorized users to select given fields from the m-Wallet to fill in values for a given input field in a wireless form.

When constructing a new mapping, the Formfiller uses name guessing heuristics to automatically suggest default values to the user. As a result, the mapping creation process is minimized, making it a "user-approved" mapping process.

Name-guessing can be done in two ways: you can enter rules for explicit mapping suggestions, (such as 'Credit Card number' to 'CreditCard:Number') or you can implement a dynamic heuristic that determines the similarities between the input field and the fields in the m-Wallet. For example, 'Deluxe user home address' would map automatically to 'Profile:Address'.

Detail Implementation and Usage

The "fixed" mapping suggestions should be placed as service parameters for the Formfiller service. The Input Parameter name should consist of 'ORACLE_ SERVICES_COMMERCE_FORMFILLER_SUGGESTIONS_' and the suggested key to use. For example, 'ORACLE_SERVICES_COMMERCE_FORMFILLER_ SUGGESTIONS_Credit Card' would be a suggested key to use. The default value must contain a valid Wallet compartment and field name. The administrator for the Formfiller should know the compartment and the field name in advance. For example:

Input Parameter Name: ORACLE_SERVICES_COMMERCE_FORMFILLER_ SUGGESTIONS_Credit Card

Default Value: CREDIT CARD:CC_NUMBER

The "dynamic" mapping suggestions are controlled by a class that implements the `GuessingHeuristic` interface. The factory method inside the `FormFillerManager` to retrieve the implementation of the guessing heuristic takes the class name from the Formfiller service parameters. The key of the property is ORACLE_SERVICES_COMMERCE_FORMFILLER_HEURISTIC.

19.6.1.3 Setting Up the Guessing Heuristics

The guessing heuristics uses keys that are defined in the service parameters for Formfiller Master Service. In order to setup, ORACLE_SERVICES_COMMERCE_ FORMFILLER_HEURISTIC defines the property that the `GuessingHeuristic` implementor of the Formfiller module uses. This value must be the fully qualified class name of the class implementing the `GuessingHeuristic` interface. This is an optional field, as the default dynamic heuristic provider is set to `oracle.panama.app.services.modules.formfiller.WalletGuessingHeuristic`.

The following are input service parameters are examples of the configuration file:

- ORACLE_SERVICES_COMMERCE_FORMFILLER_HEURISTIC

  The default value for this parameter is oracle.panama.app.services.modules.formfiller.WalletGuessingHeuristic

- ORACLE_SERVICES_COMMERCE_FORMFILLER_SUGGESTIONS_ is the prefix used by the Formfiller module to define fixed mappings for the guessing heuristics. The key must be appended to this prefix and inserted as an input parameter for the Formfiller service. It maps a key to some value. The key is matched against the label and the variable name of the input fields for the new mapping (in that order). The administrator must enter the correct values for the

keys, matching them, for example, to the Wallet fields. For example, an administrator matches the values to the keys to the wallet fields as follows: The following is an example of such a file:

ORACLE_SERVICES_COMMERCE_FORMFILLER_SUGGESTIONS_Credit Card

Default Value: CREDIT CARD:CC_NUMBER

19.6.1.4 Using the Formfiller Administration

The Formfiller Administration enables you to manage settings, manipulate stored mappings, and approve pending mappings.

To access the Formfiller Administration:

1. Select the Content Manager. The Root Folders and Services screen appears.
2. From the browse screen for the root-level folders and services, select the Commerce Folder.
3. Select Formfiller.
4. Click Edit. The Edit Service screen appears.
5. From the left panel of the Edit Service screen, select Master Service.
6. From the Edit Master Service screen, click Configure. The Formfiller Administration appears and defaults to the Config tab.

The Config tab enables you to set the submission mode for the Formfiller mappings by selecting between the following options:

- Open -- Enables all users to submit mappings.
- Closed -- Restricts all users from submitting mappings.
- Restricted -- Only selected users can submit mappings.

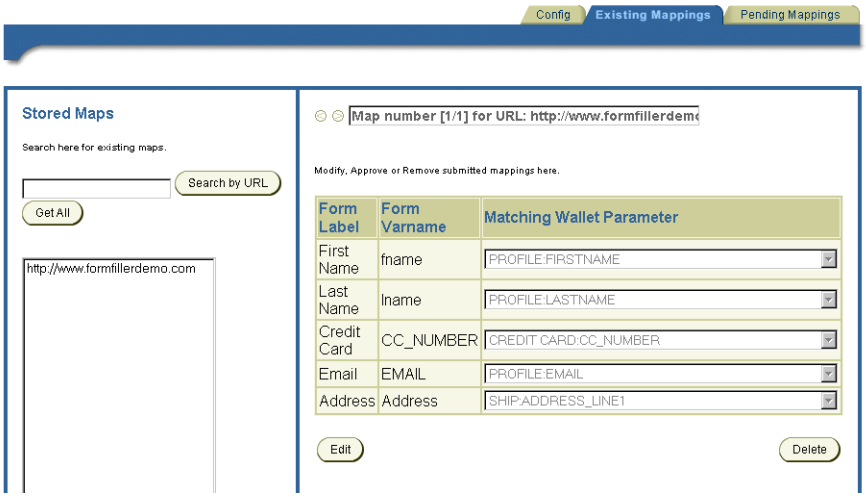
The Config tab also includes the Auto-Approve Mode. Selecting this option approves all submitted mappings immediately. (These mappings do not need approval as they become effective immediately.)

Figure 19–2 The Config Tab of the Formfiller Administration

The Existing Mappings tab enables you to search for, edit, and delete existing Formfiller mappings.

To retrieve a stored mapping, either search for the mapping by URL, or select Get All. The mapping appears in the pane in the Stored Maps section of the screen. To edit a mapping, click on the mapping. The mapping's form label, variable name (Varname) and matching wallet parameters appear in the right frame. You can then modify the mapping by using the drop-down lists to select different matching wallet parameters. Click done after you have completed your changes. Clicking Delete removes the mapping.

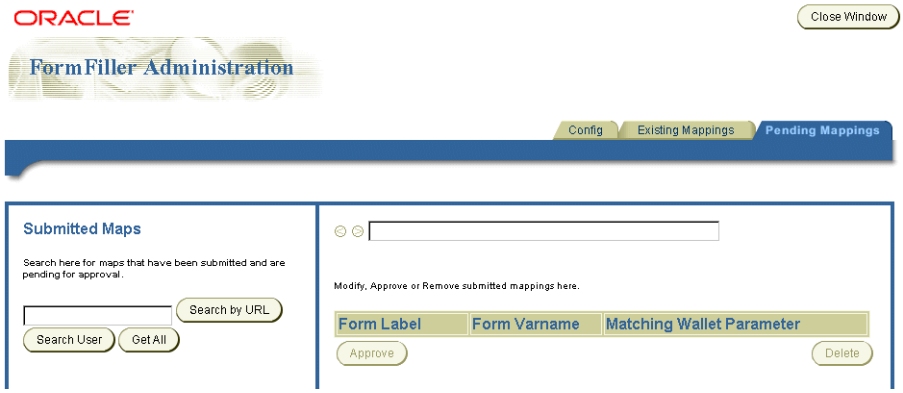
Figure 19–3 The Existing Mappings Tab of the Formfiller Administration



The Pending Mappings tab enables you to search for, edit, delete, and approve any pending (unapproved) mappings.

You can retrieve a pending mapping either by searching by URL, or by user. To retrieve all the pending mappings, select Get All. The mappings appear in the pane in the Stored Maps section of the screen. To select a mapping, click on the mapping. The mapping's form label, variable name (Varname) and matching wallet parameters appear in the right frame. You can then approve the mapping or delete it.

Figure 19–4 The Pending Mappings Tab of the Formfiller Administration



Note: For performance reasons, (such as a database connection cache with a five-minute expiration period) it can take up to five minutes for changes made using the Formfiller Administration to be reflected in the system.

19.6.1.5 Configuring the Input Parameters for the Formfiller Module

To configure the input parameters for this module:

The Formfiller module includes the following optional input parameters, which do not require configuration.

- ORACLE_SERVICES_COMMERCE_FORMFILLER_HEURISTIC
  - Description: A fully qualified classname of formfiller guessing heuristic class, if a user wants to override the default guessing implementation.
  - Valid Values: Null - to use the default guessing heuristic class package.formfiller.myGuessingHeuristic
  - Default Value: Null.
  - Customizability: The user needs to customize this value only if they want to override the default heuristic mechanism.
- ORACLE_SERVICES_COMMERCE_FORMFILLER_SUGGESTIONS_<label_key or variable_key>=<Wallet <compartment>:<field>>

- Description: A suggestion that corresponds to the label or variable in the key. Whenever formfiller receives <label_key> or <variable_key>, it automatically points to the corresponding compartment and field in Wallet. For example: ORACLE_SERVICES_COMMERCE_FORMFILLER_SUGGESTIONS_ccnum=CREDIT_CARD:CC_NUMBER

- ORACLE_SERVICES_COMMERCE_FORMFILLER_SUGGESTIONS_fn=PROFILE:FIRSTNAME
  - Default Value: Null.
  - Customizability: Users only need to customize these values if they want to add suggestions.

19.6.1.6 Linking to the Formfiller Module

You link to the Formfiller module using the following virtual URL:

omp://oracle/services/commerce/formfiller

The Formfiller module includes the following input call parameters:

Table 19–9 Input Call Parameters of the Formfiller Module

Parameter Name	Mandatory?	Description	Valid Value	Triggers Output
FORMFILLURL	Yes	The URL of the form to be filled. Restriction: URL encoded	A string. For example: FORMFILLURL=http://www.formfillerdemo.com	ReturnGroup
FORMFILLPARAMS	Yes	The parameters inside the form. Restriction: It should be a comma-separated, ordered list of [%label%:%variable name%] pairs.Where %label% is the label used in the form that is used for the %variable name% variable. The Actual parameters must be URL encoded	.A string. For example: FORMFILLPARAMS=First+Name:fname,Last+Name:lname,Credit+Card:CC_NUMBER,Email:EMAIL,Address:Address	ReturnGroup

Parameter Name	Mandatory?	Description	Valid Value	Triggers Output
APPLICATION	No	Specifies the application name to identify the request to the Formfiller (which in turn passes it to the m-Wallet). When it is not specified, the URL will be treated as the application name. This must be URL encoded.	A string. For example: APPLICATION=ForFiller Demo	ReturnGroup

19.6.1.7 Output Parameters

The Formfiller's output parameters include the following:

ReturnGroup

This group includes the following parameters, which return the values for the Formfiller.

Table 19–10 Parameters of ReturnGroup

Parameter	Mandatory	Description	Valid Values
FORMFILLURL	Yes	The URL of the form to be filled. Restriction: URL encoded	A string. For example: FORMFILLURL=http://www.formfillerdemo.com
FORMFILLPARAMS	Yes	The parameters inside the form. Restrictions: ■ The parameters must be URL encoded. ■ For successful retrievals, the parameters should be a comma-separated ordered list of [%label%:%variable name%:%value%] pairs. Where %label% is the label used in the form that is used for the %variable name% variable. %value% contains the result from the m-Wallet. ■ For unsuccessful retrievals, the parameters do not return anything for the values.	A string. For example: FORMFILLPARAMS=First Name:fname:Bob,Last Name:lname:Smith,Credit Card:CC_NUMBER:123456789,Email:EMAIL:bob.smith@company.com,Address:Address:SomeWhereOnEarth Example: FORMFILLPARAMS=First Name:fname:,Last Name:lname:,Credit Card:CC_NUMBER:,Email:EMAIL:,Address:Address:

Parameter	Mandatory	Description	Valid Values
SUCCESSCODE	Yes	The success code indicates whether there was a successful request of information from the m-Wallet for the given labels and variable names.	The valid values are: TRUE -- For successful data retrieval FALSE --For Unsuccessful retrieval of data because the user cancelled or was or unable to retrieve dynamic mapping. For example: ■ SUCCESSCODE=TRUE ■ SUCCESSCODE=FALSE

19.6.1.8 Examples

For a the successful data retrieval for the application, FormFiller Demo, configure the parameters as follows:

- Input Parameters:
  - FORMFILLURL=http://www.formfillerdemo.com
  - FORMFILLPARAMS=First+Name:fname,Last+Name:lname,Credit+Card:CC_NUMBER,Email:EMAIL,Address:Address
  - APPLICATION=FormFiller Demo
- Output Parameters:
  - FORMFILLURL=http://www.formfillerdemo.com
  - FORMFILLPARAMS=First+Name:fname:Bob,Last+Name:lname:Smith,Credit+Card:CC_NUMBER:123456789,Email:EMAIL:bob.smith@company.com,Address:Address:Some+Street+On+Earth
  - SUCCESSCODE=TRUE

An example of the unsuccessful retrieval of data for the application, FormFiller Demo, is as follows:

- Input Parameters:
  - FORMFILLURL=http://www.formfillerdemo.com
  - FORMFILLPARAMS=First+Name:fname,Last+Name:lname,Credit+Card:CC_NUMBER,Email:EMAIL,Address:Address
  - APPLICATION=FormFiller Demo

- Output Parameters:
  - FORMFILLURL=http://www.formfillerdemo.com
  - FORMFILLPARAMS=First+Name:fname:,Last+Name:lname:,Credit+Card:CC_NUMBER:,Email:EMAIL:,Address:Address:
  - SUCCESSCODE=FALSE

19.7 Creating a Billing Mechanism

Billing with Oracle9*i*AS Wireless is based on the service activity logging framework. The Activity Logger provides the logging framework used by the runtime components. Database logging is handled asynchronously because the runtime logging on the database carries a huge overhead. The runtime data is generated as files, which are less expensive. The data thus generated is picked up by the Performance Logger framework and written onto the database. When services are executed, the log tables contains a record with information of the user, timestamp, service name and the related values. Related values are values that can be associated with the service, such as a cost. The framework can be extended to create a custom cost-handling mechanism. For more information on logging activity, system logging tables and how to manage the logging system, refer to the Oracle9*i*AS Wireless Getting Started and System Guide.

20

Location-Based Module

This document describes the reusable Location-Based Module, included in Oracle9*i*AS Wireless. Each section of this document presents a different topic. These sections include:

- Section 20.1, "Location Modules"
- Section 20.2, "Driving Directions"
- Section 20.3, "The Business Directory Module"
- Section 20.4, "Maps Module"
- Section 20.5, "Extending the Mobile Modules"

20.1 Location Modules

Location-based services make mobile applications easier to use and provide them with quick access to timely and critical information.

20.1.1 Location Picker

The Location Picker module enables users to pick and manage their frequently-accessed locations. Using this module, a user can specify a location that can be used by another module, such as the driving directions module. This location can be the user's default location, the current location (if mobile positioning is enabled), a Locationmark selected by the user, a recent location used by the user, or a new location to be entered by the user

The Location Picker module is used by other modules to acquire a location from the user. When used directly by the user, Location Picker provides management of the user’s locationmarks and allows the user to set his "preferred" location, which is

either the user's current location (when mobile positioning is available and on) or the user's default locationmarks.

Other location modules include Driving Directions, Maps, and Business Directory. These modules use the Location Picker to acquire location(s) from the user if the user does not have a "preferred" location or if the user specifically wants to change the location used for those modules.

This module integrates with positioning servers when available and is built upon the Oracle9*i*AS Wireless Location Application Component API.

20.1.2 Configuring the Location Picker Module

This service requires the Oracle9*i*AS Wireless geocoding provider only when the geocoding of addresses is needed and the Oracle9*i*AS Wireless mobile positioning provider only when the positioning feature is needed. The geocoding and mobile positioning are optional features.

Table 20–1 Requirements for the Location Picker Module

Name	External Providers	Instructions	From
Geocoding Provider	otn.oracle.com	See Section 15.1.3.	2.0
Mobile positioning provider	otn.oracle.com	See Section 15.1.3.	2.0

This module does not require scripts.

20.1.2.1 Configuring the Input Parameters of the Location Picker Module

The Location Picker module includes the following input parameters:

- ORACLE_SERVICES_LOCATION_PICKER_STACKSIZE
  - Description: The depth of the location history stack. When it reaches this depth, the least recent locations are replaced by new locations to keep the depth constant.
  - Default Value: 72
- ORACLE_SERVICES_LOCATION_PICKER_RECS_PER_PAGE
  - Description: The number of locations displayed per page. Used to indicate how many locations are displayed per page.
  - Default Value: 9

20.1.2.2 Linking to the Location Picker Module

You link to the location picker module using the following virtual URL:

omp:/ / oracle / services / location / picker - Invocation Interface

Input Parameters

The location picker has the following input call parameters:

- LOCATIONTITLE
  - Description: The name of the location to be specified. This will be displayed throughout the Location Picker module as the title of the page.
  - Valid Value: A String. For example:
    - LOCATIONTITLE=Map
    - LOCATIONTITLE=Destination Location
- LOCATIONQUALITY
  - Mandatory: No
  - Description: The quality of the location to be specified. This will be used to check if the specified location meets the required quality.
  - Valid Values:
    - 1 (Address quality)
    - 2 (Street quality)
    - 3 (Intersection quality)
    - 4 (Postalcode quality)
    - 5 (City quality)
    - 6 (County quality)
    - 7 (State quality)
    - 8 (Country quality)
    - 11 (Unknown quality)
- LOCATIONMASK
  - Mandatory: No

- Description: The mask used to specify which location fields will be available when entering a new location.
- Valid Value: An integer derived by bitwise ORing together the integer values for all the location fields wanted. The values are defined as follows:
  - COMPANYNAME_FIELD = 1
  - FIRSTLINE_FIELD = 2
  - SECONDLINE_FIELD = 4
  - LASTLINE_FIELD = 8
  - BLOCK_FIELD = 16
  - CITY_FIELD = 32
  - COUNTY_FIELD = 64
  - STATE_FIELD = 128
  - COUNTRY_FIELD = 256
  - POSTALCODE_FIELD = 512
  - POSTALCODEEXT_FIELD = 1024
  - LAT_FIELD = 2048
  - LNG_FIELD = 4096
  - Examples:

    LOCATIONMASK=14 (address line 1, address line 2, address last line)
    LOCATIONMASK=162 (address line 1, city, state)

- MOD
  - Mandatory: Optional
  - Description: This parameter is used to specify a condition on the returned location. For example, if the user only wants to choose among the named location (for example, Location Marks), then use MOD="LM". If unspecified, the default mode will be used (for example, all available locations will be offered as choices).
  - Valid Value: LM (Allows to choose among existing Location Marks or create new ones.)

    Example: MOD=LM

Output Parameters

The output parameters for the Location Picker module include the following:

Table 20–2 Output Parameters of the Location Picker

Parameter Name	Mandatory	Description	Valid Value
CN	No	Company Name	A string. For example: CN=Oracle Corp.
FL	No	Address First Line	A string. For example: FL=500 Oracle Parkway
SL	No	Address Second Line	A string. For example: SL=Redwood City, CA
LL	No	Address Last Line	A string. For example: LL=US
BL	No	Block	A string. For example: BL=Block 400
CI	No	City	A string. For example: CI=Redwood City
CT	No	County	A string. For example: CT=San Mateo
ST	No	State	A string. For example: ST=CA
PC	No		Postal Code A string. For example: PC=94065
PCE	No	Postal Code Extension	A string. For example: PCE=5423
CO	No	Country	A string. For example: CO=US
LT	No	Latitude	Double. For example: LT=37.2433
LN	No	Longitude	Double. For example: LN=-122.3452
N	No	Name	A string. For example: N=Golden Gate Park
LMN	No	Location Mark Name	A string. For example: LMN=Office
STATUS	No	The status of the module call.	(Ok) CANCEL (Cancelled)

20.2 Driving Directions

The Driving Directions module allows a mobile application to provide its users with driving directions between an originating address and a destination address. It links to the Location Picker module to enable users to select originating and destination addresses not provided by Driving Directions. The Driving Directions module also links with the Maps module for enhanced routing.

This module is built upon the Oracle9*i*AS Wireless Location Application Component API.

20.2.1 Configuring the Driving Directions

This service requires the Oracle9*i*AS Wireless routing provider.

Table 20–3 Requirements for the Location Picker Module

Name	External Provider(s)	Instructions	From
Routing Provider	otn.oracle.com	See Section 15.1.3.	2.0

This module does not require scripts.

20.2.1.1 Input Parameters

The Driving Directions module includes the following input parameters:

- ORACLE_SERVICES_LOCATION_ROUTER_RECS_PER_PAGE
  - Description: Items per page. Used to indicate how many steps are displayed per page.
  - Default Value: 9
- ORACLE_SERVICES_LOCATION_ROUTER_VOICETOKENFILE
  - Description: The absolute file path for the voice token file--a properties file that gives the mapping between recorded **.wav** files and voice outputs.
  - Default Value: .

20.2.1.2 Linking to the Driving Directions Module

You link to the Driving Directions module through the following virtual URL:

omp://oracle/services/location/directions

Input Call Parameters

The Driving Directions module includes the following input call parameters:

Table 20–4 Input Call Parameters of the Driving Directions Module

Parameter Name	Mandatory	Description	Valid Value
OCOMPANYNAME	No	Company Name of Starting Location	A string. For example: OCOMPANYNAME=Oracle Corp.
OADDRESS	No	Address First Line of Starting Location	A string. For example: OADDRESS=500 Oracle Parkway
OADDRESS2	No	Address Second Line of Starting Location	A string. For example: OADDRESS2=Redwood City, CA
OADDRESSLL	No	Address Last Line of Starting Location	A string. For example: OADDRESSLL=US
OBLOCK	No	Block of Starting Location	A string. For example: OBLOCK=Block 400
OCITY	No	City of Starting Location	A string. For example: OCITY=Redwood City
OCOUNTY	No	County of Starting Location	A string. For example: OCOUNTY=San Mateo
OSTATE	No	State of Starting Location	A string. For example: OSTATE=CA
OZIP	No	Postal Code of Starting Location	A string. For example: OZIP=94065
OZIPEXT	No	Postal Code Extension of Starting Location	A string. For example: OZIPEXT=5423
OCOUNTRY	No	Country of Starting Location	A string. For example: OCOUNTRY=US
OLAT	No	Latitude of Starting Location	(Double) For example: OLAT=37.2433
OLNG	No	Longitude of Starting Location	(Double) For example: OLNG=-122.3452
ONAME	No	Name of Starting Location	A string. For example: ONAME=Golden Gate Park

Table 20–4 Input Call Parameters of the Driving Directions Module

Parameter Name	Mandatory	Description	Valid Value
DCOMPANYNAME	No	Company Name of Destination Location	A string. For example: DCOMPANYNAME=Oracle Corp.
DADDRESS	No	Address First Line of Destination Location	A string. For example: DADDRESS=500 Oracle Parkway
DADDRESS2	No	Address Second Line of Destination Location	A string. For example: DADDRESS2=Redwood City, CA
DADDRESSLL	No	Address Last Line of Destination Location	A string. For example: DADDRESSLL=US
DBLOCK	No	Block of Destination Location	A string. For example: DBLOCK=Block 400
DCITY	No	City of Destination Location	A string. For example: DCITY=Redwood City
DCOUNTY	No	County of Destination Location	A string. For example: DCOUNTY=San Mateo
DSTATE	No	State of Destination Location	A string. For example: DSTATE=CA
DZIP	No	Postal Code of Destination Location	A string. For example: DZIP=94065
DZIPEXT	No	Postal Code Extension of Destination Location	A string. For example: DZIPEXT=5423
DCOUNTRY	No	Country of Destination Location	A string. For example: DCOUNTRY=US
DLAT	No	Latitude of Destination Location	(Double) For example: DLAT=37.2433
DLNG	No	LLongitude of Destination Location	(Double) For example: DLNG=-122.3452
DNAME	No	Name of Destination Location	A string. For example: DNAME=Golden Gate Park

Output Parameters

The Driving Directions Module includes the following output parameters:

Table 20–5 Output Parameter of the Driving Directions Module

Parameter Name	Mandatory	Description	Valid Value
STATUS	No	The status of a mobile call.	(OK) CANCEL (Cancelled)

20.3 The Business Directory Module

The Business Directory module provides user with a complete business directory. This module is built on the Oracle9*i*AS Wireless Location Application Component API.

This module provides a "yellow pages" type interface to look for the addresses and phone numbers of registered businesses in a given radius. It has search capabilities for business names or categories. Browsing through categories is also enabled. If no location parameters are passed to this module, the location module is invoked to obtain location data for the search.

20.3.1 Configuring the Business Directory Input Parameter

This module requires the Oracle9*i*AS Wireless business directory provider.

Table 20–6 Requirements for the Business Directory Module

Name	External Provider(s)	Instructions	From
Business Directory Provider	otn.oracle.com	See Section 15.1.3.	2.0

This module does not require scripts.

20.3.1.1 Configuring the Input Parameters

The Business Directory includes the following input parameters:

- ORACLE_SERVICES_LOCATION_BIZDIR_RECS_PER_PAGE
  - Description: Items per page. Used to indicate how many businesses/categories are displayed per page.
  - Default Value: 9

20.3.1.2 Linking to the Business Directory

You link to the Business Directory Module using the following virtual URL:

omp://oracle/services/location/bizdir

The Business Directory Module includes the following input call parameters:

Table 20–7 The Input Call Parameters of the Business Directory Module

Parameter Name	Mandatory	Description	Valid Value
PH	No	Phrase (keywords) to search for.	A string. For example: PH=Pizza PH=Restaurants PH=Oracle
FC	No	Full category of the business. This category is defined in the YP mapping XML file, which is specified using the Oracle9*i*AS Wireless Webtool.	A string. For example: FC=/Business/Restaurant/Italian For example: FC=/Business/Automotive/Dealer/New/BMW
CN	No	Company Name	A string. For example: CN=Oracle Corp.
FL	No	Address First Line	A string. For example: FL=500 Oracle Parkway
SL	No	Address Second Line	A string. For example: SL=Redwood City, CA
LL	No	Address Last Line	A string. For example: LL=US
BL	No	Block	A string. For example: BL=Block 400
CI	No	City	A string. For example: CI=Redwood City
CT	No	County	A string. For example: CT=San Mateo
ST	No	State	A string. For example: ST=CA
PC	No	Postal Code	A string.For example: PC=94065

Table 20–7 The Input Call Parameters of the Business Directory Module

Parameter Name	Mandatory	Description	Valid Value
PCE	No	Postal Code Extension	A string. For example: PCE=5423
CO	No	Country	A string. For example: CO=US
LT	No	Latitude	(Double) For example: LT=37.2433
LN	No	Longitude	(Double) For example: LN=-122.3452
N	No	Name	A string. For example: N=Golden Gate Park

Output Parameter

The Driving Directions Module includes the following output parameter:

Table 20–8 Output Parameter of the Driving Directions Module

Parameter Name	Mandatory	Description	Valid Value
STATUS	No	The status of a mobile call.	(OK) CANCEL (Cancelled)

20.4 Maps Module

The Maps module provides broad and detailed maps for a given location, supports map tiling and image map transformation for different devices. This module integrates with the Driving Directions module and is built upon the Oracle9*i*AS Wireless Location Application Component API.

20.4.1 Configuring the Maps Input Parameters

This service requires the Oracle9*i*AS Wireless mapping provider.

Table 20–9 Requirements for the Maps Module

Name	External Providers	Instructions	From
Mapping Provider	otn.oracle.com	See Section 15.1.3.	2.0

20.4.2 Configuring the Input Parameters

The Maps module includes the following input parameters:

- ORACLE_SERVICES_LOCATION_MAPS_WIDTH_MS
  - Description: The width of maps on PDAs and PC browsers.
  - Default Value: 200
- ORACLE_SERVICES_LOCATION_MAPS_HEIGHT_MS
  - Description: The height of maps on PDAs and PC browsers.
  - Default Value: 200

20.4.3 Linking to the Maps Module

You link to the Maps module using the following virtual URL:

omp://oracle/services/location/maps

Input Call Parameters

The Maps module includes the following input call parameters:

Table 20–10 Input Call Parameters of the Maps Module

Parameter Name	Mandatory	Description	Valid Value
CN	No	Company Name	A string. For example: CN=Oracle Corp.
FL	No	Address First Line	A string. For example: FL=500 Oracle Parkway
SL	No	Address Second Line	A string. For example: SL=Redwood City, CA
LL	No	Address Last Line	A string. For example: LL=US
BL	No	Block	A string. For example: BL=Block 400
CI	No	City	A string. For example: CI=Redwood City
CT	No	County	A string. For example: CT=San Mateo
ST	No	State	A string. For example: ST=CA

Table 20–10 Input Call Parameters of the Maps Module

Parameter Name	Mandatory	Description	Valid Value
CN	No	Company Name	A string. For example: CN=Oracle Corp.
PC	No		Postal Code A string. For example: PC=94065
PCE	No	Postal Code Extension	A string. For example: PCE=5423
CO	No	Country	A string. For example: CO=US
LT	No	Latitude	Double. For example: LT=37.2433
LN	No	Longitude	Double. For example: LN=-122.3452
N	No	Name	A string. For example: N=Golden Gate Park
LMN	No	Location Mark Name	A string. For e xample: LMN=Office
STATUS	No	The status of the module call.	(Ok) CANCEL (Cancelled)

Output Parameter of the Maps Module

The Maps module includes the following output parameter

Table 20–11 Output Parameter of the Map Module

Parameter Name	Mandatory	Description	Valid Value
STATUS	No	The status of a mobile call.	(OK) CANCEL (Cancelled)

20.5 Extending the Mobile Modules

The location modules use the Oracle9*i*AS Wireless Location Application Component APIs. These Java APIs can be used independently to write other Location Based Services (LBS). For more information on the Location Application Component API, see Section 15.2.

This section describes the main classes of these APIs. For each class, the typical use cases and the code example for each use case are given.

20.5.1 The oracle.panama.model.LocationMark class

This interface represents a Location Mark saved by the user. It extends the Location interface which represents a location.

The typical operations a user wants to perform include:

1. Get all Location Marks of a user

   ```
   LocationMark[ ] locationmarks = iasUser.getLocationMarks();
   ```
   where iasUser is an object of User, which represents a user of Oracle9*i*AS Wireless wireless.

2. Get and set attribute(s) of a Location Mark The LocationMark class has specific "get" and "set" methods. For example, to get the city of a Location Mark, use String city = locationmark.getCity();

   To set the city, use

   ```
   locationmark.setCity(city);
   ```

   For a complete listing of these methods, refer to the Oracle9*i*AS Wireless javadoc.

3. Create a Location Mark

   ```
   String locationmark_name = "Office";
   LocationMark lm =
   MetaLocator.getInstance().getModelFactory().createLocationMark(locationmark_
   name,iasUser);
   Location l = SpatialManager.createLocation("Oracle", "500 Oracle
   Parkway","", "Redwood City, CA 94065", "US");
   lm.setLocation(l);
   Locator.getInstance().getPersistentLocator().getSessionManager().commitSessi
   on();
   ```

4. Delete a Location Mark

   ```
   LocationMark lm =
   MetaLocator.getInstance().getModelServices().lookupLocationMark(locationmark
   _id);
   lm.delete();
   MetaLocator.getInstance().getModelFactory().save();
   ```

   where `locationmark_id` is an automatically generated unique Java "long" which identifies the Location Mark. To get this id, call `getId()` on the LocationMark object.

5. Update a Location Mark The steps to update a Location Mark is similar to creating a new one. First, get the Location Mark object. Second, update the location. Finally, commit the changes.

```
LocationMark lm =
MetaLocator.getInstance().getModelServices().lookupLocationMark(locationmark
_id);
lm.setCity("San Francisco");
Locator.getInstance().getPersistentLocator().getSessionManager().commitSessi
on();
```

6. Set a Location Mark as the default location

```
LocationMark lm =
MetaLocator.getInstance().getModelServices().lookupLocationMark(locationmark
_id);
iasUser.setDefaultLocationMark(lm);
MetaLocator.getInstance().getModelFactory().save();
```

20.5.2 The oracle.panama.spatial.geocoder.Geocoder class

The Geocoder interface abstracts the implementation of a geocoder. The geocoder implementation is specific to the geocoding content provider. For example, to use the geocoding content from MapQuest, the implementation class is `oracle.panama.spatial.core.geocoder.GeocoderImplMapQuest`, which is supplied with Oracle9*i*AS Wireless. At the module level, the specific implementation class is transparent.

The typical operations a user wants to perform include:

1. Get an instance of Geocoder

```
Geocoder geocoder = SpatialManager.getGeocoder();
```

2. Geocode a location. There are two ways:

   - Use the Geocoder interface:

     ```
     Location[} locations = geocoder.geocodeAddress(location, matchmode);
     ```

     Where location is a Location object and matchmode is a String indicating how the geocoder should try to geocode the location. The valid values for matchmode are:

  - MATCH_MODE_NORMAL, MATCH_MODE_STANDARD, MATCH_MODE_TIGHT
  - MATCH_MODE_RELAXED

  Refer to the Oracle9*i*AS Wireless javadoc for details.

- Use the Location interface to geocode a location.

  ```
  location.geocode(checkWhetherNecessary,
  makeCorrections);
  ```

  where check`WhetherNecessary` and `makeCorrections` are two booleans.

20.5.3 The oracle.panama.module.location.LocationHistoryManager class

The `LocationHistoryManager` class manages the locations saved automatically by other modules or applications. It has methods for:

1. Get all automatically saved locations (that is, location history).

   ```
   Location[] locations = LocationHistoryManager.getLocations(iasUser);
   ```

   This method returns an array of locations sorted by their last used time stamps.

2. Update the location history with a location.

   The location history is a stack of locations saved by modules or applications. The history can be used by modules and applications to eliminate the need for users to enter those locations again. The history also can be shared among all location based services. For example, a restaurant found using the Business Directory module is saved in the history and is be automatically available when the user uses the Driving Directions module to get directions to that restaurant. The maximum size of the stack is configurable by the ORACLE_SERVICES_LOCATION_PICKER_STACKSIZE service parameter. When the stack size is below the maximum, new locations are added to the stack. When the stack size reaches the maximum, a new location replaces the earliest location on the stack (that is, the location with the earliest time stamp).

   ```
   long id =LocationHistoryManager.updateHistory(l, iasUser);
   ```

Where `l` is an object of `LocationHelper`, a class that implements the Location interface. The above code updates the location history stack with location "l" for `iasUser`. Refer to Oracle9*i*AS Wireless modules javadoc for details on the `LocationHelper` class.

20.5.4 The oracle.panama.spatial.router.Router class

The Router interface abstracts the implementation of a router, which provides turn-by-turn driving directions. The router implementation is specific to the routing content provider. For example, to use the routing content from MapQuest, the implementation class will be `oracle.panama.spatial.core.router.RouterImplMapQuest`, which is supplied with Oracle9*i*AS Wireless. At the module level, the specific implementation class used is transparent. The typical operations a user wants to perform include:

1. Get an instance of Router.

```
Router router = SpatialManager.getRouter();
```

2. Get Driving Directions between two locations.

```
RoutingResult rr = router.computeRoute(starting_location,    destination_
location, via_points, settings, locale);
```

where `starting_location` and `destination_location` are two Location objects, `via_points` is an array of Location objects that represent the desired via points along a route (null is okay if no via points is needed), settings is a `RoutingSettings` object that indicates the routing options (see below for details), and locale is the desired locale for directions output (for example, mile is used as distance unit for United States, whereas km is used in Canada). Once the `RoutingResult` object is obtained, you can iterate through the maneuvers to get turn-by-turn directions. For example:

```
for (int i = 0; i < rr.getManeuvers().length; i++){
  Maneuver m = rr.getManeuvers()[i];
  System.out.println(m.getNarrative());
}
```

3. Set Driving Directions options Use RoutingSettings class to set options. For example, to get the overview map of a route:

```
RoutingSettings settings = new RoutingSettings();
```

```
settings.setRequestMap(true);
settings.setSecondaryOption(RoutingOption.overviewMapWidth, MAP_WIDTH+"");
settings.setSecondaryOption(RoutingOption.overviewMapHeight, MAP_HEIGHT+"");
RoutingResult rr = router.computeRoute(starting_location, destination_
location, via_points, settings, locale);
String url = rr.getOverviewMapURL();
```

Refer to Oracle9*i*AS Wireless javadoc for details of available options.

20.5.5 The oracle.panama.spatial.mapper.Mapper class

The Mapper interface abstracts the implementation of a mapper, which provides maps. The mapper implementation is specific to the mapping content provider. For example, to use the mapping content from MapQuest, the implementation class will be `oracle.panama.spatial.core.mapper.MapperImplMapQuest`, which is supplied with Oracle9*i*AS Wireless. At the module level, the specific implementation class used is transparent. The typical operations a user wants to perform include:

1. Get an instance of Mapper.

   ```
   Mapper mapper = SpatialManager.getMapper();
   ```

2. Get the map of a location There are several methods available in the Mapper class. For example:

   ```
   String mapurl = mapper.getMapURL(location, ImageFormats.GIF,
     location.getLongitude() - zoomlevel * MAP_LNG_SIZE,
     location.getLongitude() + zoomlevel * MAP_LNG_SIZE,
     location.getLatitude() - zoomlevel * MAP_LAT_SIZE,
     location.getLatitude() + zoomlevel * MAP_LAT_SIZE,
     MAP_WIDTH, MAP_HEIGHT,
     false);
   ```

   Refer to the Oracle9*i*AS Wireless javadoc for details of this and other mapping methods.

20.5.5.1 The oracle.panama.spatial.yp.YPFinder class

The `YPFinder` interface abstracts the implementation of a "YP" like business directory service. The `YPFinder` implementation is specific to the business directory content provider. For example, to use the content from MapQuest, the implementation class will be `oracle.panama.spatial.core.yp.YPFinderImplMapquest`, which is

supplied with Oracle9*i*AS Wireless. At the module level, the specific implementation class used is transparent. The typical operations a user wants to perform include:

1. Get an instance of YPFinder.

   ```
   YPFinder finder = SpatialManager.getYPFinder();
   ```

2. Get business categories The `YPFinder` class supports the "browsing" of business categories. These categories are defined by user based on the content provider in oracle.panama.spatial.yp.**YPCategories.xml** file. The location of this file needs to be specified in Oracle9*i*AS Wireless webtool. At runtime, the `YPFinder` class has methods that support the browsing of the category hierarchy. For example, to start at the root category,

   ```
   YPCategory root = finder.getCategoryAtRoot();
   ```

   Then to get the subcategories at root,

   ```
   YPCategory[] cats = root.getSubCategories();
   ```

3. Search for businesses. The "search" operation can be performed with two classes: `YPFinder` and `YPCategory`. The search can be proximity search (that is, around a location within a certain radius), using such criteria as searching a city, or a zipcode.

   ```
   YPBusiness[] bizes = root.getBusinessesInRadius(location, raidus_in_meters,
   locale);
   YPBusiness[] bizes = finder.getBusinessesInSameCity(bizname, location,
   locale);
   YPBusiness represents a business and extends the Location class. So you can
   get the address of the business using this class or use getTelephone() to
   get the business phone.
   ```

20.5.5.2 Location Mark API Examples

This example creates a Location Mark, "Office". Note that the call `SpatialManager.createLocation()` automatically geocodes the location.

Source

```
import oracle.panama.model.LocationMark;
import oracle.panama.model.Location;
import oracle.panama.model.MetaLocator;
```

```
import oracle.panama.core.util.Locator;
import oracle.panama.spatial.SpatialManager;
String locationmark_name = "Office";
LocationMark lm =
MetaLocator.getInstance().getModelFactory().createLocationMark(locationmark_
name,iasUser);
Location l = SpatialManager.createLocation("Oracle","500 Oracle Parkway","",
"Redwood City, CA 94065","US");
lm.setLocation(l);
Locator.getInstance().getPersistentLocator().getSessionManager().commitSession()
;
```

20.5.5.3 Driving Directions API Examples

This section provides an example of the Driving Directions API and its output.

Source

```
import oracle.panama.model.LocationMark;
import oracle.panama.model.Location;
import oracle.panama.spatial.SpatialManager;
import oracle.panama.spatial.router.Router;
import oracle.panama.spatial.router.RoutingSettings;
import oracle.panama.spatial.router.RoutingOption;
import oracle.panama.spatial.router.RoutingResult;
import oracle.panama.spatial.router.Maneuver;
Location starting= SpatialManager.createLocation("Oracle","500 Oracle
Parkway","", "Redwood City, CA 94065","US");
Location destination = SpatialManager.createLocation("Autobahn Motors","700
Island Pkwy","", "Belmont, CA","US");
Router r = SpatialManager.getRouter();
RoutingSettings settings = new RoutingSettings();
settings.setRequestMap(true);
settings.setSecondaryOption(RoutingOption.overviewMapWidth, "400");
settings.setSecondaryOption(RoutingOption.overviewMapHeight, "400");
RoutingResult rr = r.computeRoute(starting, destination, null, settings,
Locale.getDefault())
for (int i = 0; i < rr.getManeuvers().length; i++){
  Maneuver m = rr.getManeuvers()[i];
  System.out.println((i+1)+")"+m.getNarrative());
}
```

Output

1. Begin at 500 Oracle Pkwy on Oracle Pkwy and go West for 0.6 miles (0.6Miles)
2. Bear right on Island Pkwy and go West for 0.3 miles (0.3Miles)
3. Make U-turn on Island Pkwy and go South for 80 feet to Island Pkwy (0.0Miles)

20.5.5.4 Maps API Examples

Source

```
import oracle.panama.model.LocationMark;
import oracle.panama.model.Location;
import oracle.panama.spatial.SpatialManager;
import oracle.panama.spatial.mapper.Mapper;
import oracle.panama.imagex.ImageFormats;
Location location= SpatialManager.createLocation("Oracle","500 Oracle
Parkway","", "Redwood City, CA 94065","US");

Mapper mapper = SpatialManager.getMapper();
String mapurl = mapper.getMapURL(location,
  ImageFormats.GIF,
  location.getLongitude() - 0.008,
  location.getLongitude() + 0.008,
  location.getLatitude() - 0.008,
  location.getLatitude() + 0.008,
  400,
  400,
  false);
```

20.5.5.5 YPFinder API Examples

This example searches for Pizza Hut within two miles of Oracle

Source

```
import oracle.panama.model.LocationMark;
import oracle.panama.model.Location;
import oracle.panama.spatial.SpatialManager;
import oracle.panama.yp.YPFinder;
import oracle.panama.yp.YPBusiness;
Location location= SpatialManager.createLocation("Oracle","500 Oracle
Parkway","", "Redwood City, CA 94065","US");
```

```
YPFinder yp = SpatialManager.getYPFinder();
YPBusiness[] bizes = yp.getBusinessesInRadius("pizza hut", location, 2*1609.344,
null);
for (int k = 0; k < bizes.length; k++) {
  YPBusiness biz = (YPBusiness) bizes[k];
  double dist = biz.getDistance(location);
  bizname = biz.getCompanyName();
  bizaddr = biz.getAddressLine1();
  bizcity = biz.getCity();
  bizstate = biz.getState();
  bizcountry = biz.getCountry();
  bizphone = biz.getTelephone();
  // output results
  //...

}
```

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y