Sun Java™ System

Portal Server Mobile Access 6 Developer's Guide

2004Q2

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

Part No: 817-6258-10

Contents

About This Guide

Sun Java™ System Portal Server Mobile Access (formerly Sun™ ONE Portal Server, Mobile Access) software is an extension to Sun Java System Portal Server (formerly Sun™ ONE Portal Server) software that enables users to access portal services from mobile devices such as cellular phones and PDAs. Mobile Access ships with three mobile applications: Calendar, Address Book, and Mail, all of which can be fully customized by developers.

The three mobile applications are based on JavaServer™ Pages (JSP™) technology, and as such, make extensive use of custom tag libraries that implement their behavior. Each tag library is customizable through a corresponding set of APIs written in the Java™ programming language.

In addition, Mobile Access software contains support for voice accessibility, which allows you to write and use applications that respond directly to voice commands.

This preface includes the following sections:

Who Should Read This Book

This guide is intended for developers who want to:

- Gain a developer's perspective of the mobile address book, calendar, and mail applications
- Create voice-enabled applications accessible from any telephone
- Change the default behavior of the application-specific tag libraries provided in this release
- Customize or extend the Portal Desktop for their mobile devices
- Deploy online help for the mobile Portal Desktop

What You Need to Know

What you need to know will vary with the characteristics of your site. Knowledge of the following is recommended:

- Sun Java Enterprise System 2004Q2
  - Sun Java System Portal Server 6 2004Q2 software
  - Sun Java System Portal Server Secure Remote Access 6 2004Q2 software (formerly Sun™ ONE Portal Server, Secure Remote Access software)
  - Sun Java System Identity Server 6 2004Q2 software (formerly Sun™ ONE Identity Server software)
  - Sun Java System Directory Server 5.2 software (formerly Sun™ ONE Directory Server software)
- Your Web Container
  - Sun Java System Application Server Standard Edition 7 Update 3 software (also known as Sun™ ONE Application Server 7 Update 3 software)
  - Sun Java System Web Server 6.1 Service Pack 1 software (Also known as Sun™ ONE Web Server 6.1 Service Pack 1 software)
- Programming and Markup Languages
  - The Java™ Programming Language
  - JavaScript™

    - Markup languages used to create portal content appropriate for users' mobile and voice environments, such as Wireless Markup Language (WML), compact Hypertext Markup Language (cHTML), Handheld Device Markup Language (HDML), Extensible Hypertext Markup Language (XHTML), and Voice Extensible Markup Language (VoiceXML).
- Other Technologies
    - Java Server Pages™ (JSP)
    - Java Servlet Technology
    - LDAP (Lightweight Directory Access Protocol)
- Your Operating System
    - Solaris™ 8 Operating System
    - Solaris™ 9 Operating System
    - Solaris™ Operating System (x 86 Platform Edition)

In addition, you should have knowledge of basic UNIX® administrative procedures and you should have knowledge of the applications provided to your users; for example, Sun Java™ System Calendar Server 6 (formerly Sun™ ONE Calendar Server).

How This Book Is Organized

This book contains the following chapters and appendixes:

- Chapter 1, Understanding the Mobile Applications, provides a developer's perspective of each mobile application by illustrating control flow through the various JSP pages.
- Chapter 2, Developing Voice Applications, explains how to add voice support to the applications that you develop. The chapter describes voice application prerequisites, and instructions for creating new voice-enabled applications.
- Chapter 3, Using the Mobile Access Public APIs, describes how to change the behavior of the application-specific tag libraries by extending the Tag Library API. The chapter also lists the utility classes exposed by the Desktop and Rendered Desktop APIs.
- Chapter 4, Deploying Online Help for the Mobile Portal Desktop, outlines the process for deploying online help files to the mobile Portal Desktop.

An index and a glossary are also provided.

Conventions Used in This Guide

The guide uses certain typographical conventions to represent types of information presented.

Monospaced Font

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

Italicized Font

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

Where to Find Related Information

In addition to this guide, this Mobile Access software release provides supplementary information for administrators as well as documentation for developers.

Use the following URL to view all the Portal Server documentation:

`http://docs.sun.com/coll/PortalServer_04q2`

Additional documents that are available include:

- *Portal Server Mobile Access Administration Guide*
- *Portal Server Mobile Access Deployment Planning Guide*
- *Portal Server Mobile Access Developer's Reference*
- *Portal Server Mobile Access Tag Library Reference*

Where to Find This Guide Online

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

http://docs.sun.com/doc/817-6258

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. Use the web-based form to provide feedback to Sun:

http://www.sun.com/hwdocs/feedback

Please provide the full document title and part number in the appropriate fields. The part number of this guide is 817-6258-10.

Chapter 1

Understanding the Mobile Applications

This chapter describes the three mobile applications provided by the Sun Java™ System Portal Server Mobile Access software (formerly known as Sun™ ONE Portal Server, Mobile Access) software. This chapter contains the following sections:

Mobile Applications Overview

Mobile Access software contains three mobile applications:

- Calendar
- Address book
- Mail

The goal of this chapter is to demonstrate the key functionality offered by these three applications, while simultaneously providing a developer's perspective of control flow through the individual JSP files.

NOTE Due to the large number of source files, this chapter does not attempt to provide a complete code review of every individual file. Instead, this chapter uses a tutorial-style format to bring users through each application, describing the key features of certain JSP files as they pertain to the current screen.

Each mobile application is based on JavaServer Pages™ (JSP™) technology, and can be accessed through any mobile device supporting any of a number of standard markup languages -- for example, WML and XHTML. These three applications are accessed through the mobile Portal Desktop. Users can select an application for use and return to the mobile Portal Desktop when finished.

For your reference, the complete JSP file list for each application is provided in control flow reference charts in the *Portal Server Mobile Access Developer's Reference*.

Each application contains at least two full sets of JSP files: an AML-based set, and a WML-based set. Most mobile devices are supported using the AML-based JSP files. The output of the AML pages is processed by the rendering engine, which translates AML to device specific markup. Some WML devices are supported by the WML-based JSP files, whose output is returned directly to the WML device.

The JSP files involved with these three applications make extensive use of five custom tag libraries: *cal*, *socs*, *ab*, *mail*, and *util*. Each tag library is fully documented, with code examples, in the *Portal Server Mobile Access Tag Library Reference.*

Preview of the util Tag Library

Three navigational tags apply to virtually every JSP file in the three sample applications: `<util:url>`, `<util:link>`, and `<util:forward>`. A fourth tag, `<util:include>`, affects the visible content on pages.

NOTE Specific resource lookup rules apply when the `comp` and `file` attributes are used in conjunction with the `url`, `include`, and `forward` tags. The `comp` and `file` attributes are parameters to the lookup process. See the Utility Tag Library (`util`) in the *Portal Server Mobile Access Tag Library Reference* for more information on these resource lookup rules.

<util:url>

This tag facilitates the construction of context-sensitive URL values. This tag is used in both the AML-based and WML-based versions of all three mobile applications.

```
<util:url file="someFile.jsp" comp="cal"/>
```

In this example, the URL specifies a file named `someFile.jsp`, and associates the file with the mobile calendar application.

NOTE This tag generates a compressed URL, which can complicate the process of looking at the generated markup to discern the target. Developers wishing to debug the application JSP pages might wish to disable URL compression in the following manner:

Edit the file:

`/etc/opt/SUNWps/MAConfig.properties`

and set the `ma.compressor.enable` property to `false`.

<util:link>

This tag facilitates the construction of link-type tags (in this guide, link-type items on a mobile device's browser are referred to as *options*), whose content is dynamically generated. Each tag contains a `tagstart` and `tagend` attribute for specifying what appears at the start and end of the generated option:

```
<util:link tagstart="<a" tagend=">">
  <util:attr attr="href">
    <util:url file="someFile.jsp" comp="cal">
... other parameters...
    </util:url>
  </util:attr>
</util:link>
```

<util:forward>

The `forward` tag forwards the request to the specified page:

```
<util:forward comp="cal" file="calHome.jsp">
```

The comp attribute associates the named file with a particular mobile application. In the preceding case, cal is an abbreviation for “calendar”.

<util:include>

This tag performs an include operation, similar to <jsp:include>, but constructs the resource name in one of two ways: with a path attribute, or with a combination of file and comp attributes. Certain JSP files use this tag to include reusable GUI elements in their output.

For more information regarding <util:url>, <util:link>, <util:forward>, and <util:include>, consult the *Portal Server Mobile Access Tag Library Reference*.

The Mobile Calendar Application

The calendar application is a useful organization tool. The application allows users to add, edit, and remove events and tasks, and includes support for repeating events, email reminders, and multiple calendar servers.

To launch a calendar application, users first log into the sample portal provided with Mobile Access. The mobile Portal Desktop displays a list of numbered options to select: User Information, Bookmarks, Personal Notes, Calendar, Address Book, and Mail.

The calendar option provides users access to the calendar server. The calendar home page summarizes the events and tasks scheduled for the day. When users select the View Calendar option, the calendar application launches, and displays the day view page, where user interaction actually begins.

NOTE Three different calendar servers are capable of interacting with this application: Sun Java System Calendar Server (formerly Sun™ ONE Calendar Server), Microsoft Exchange, and Lotus Notes. Three different sets of JSP pages are used to support all the features made available by these three servers. A generic set of JSP pages supports features common to all three calendar servers. In fact, the generic set covers all the Microsoft Exchange features. However, the Java System and Lotus Notes servers have unique features found respectively in the cal/sun-one, and cal/notes directories.

The Day View Page

The name of the file that displays the day view page is `dayview.jsp`. On a browser, this page displays the current calendar ID, the date, and a summary of any tasks or events currently scheduled. The displayed page also provides options that users can select in order to add a new event, add a new task, view other calendars, or reset the calendar to a specific date. In the source code, such options are specified as `<util:link>` from the util tag library, as described in "<util:link>" on page 13.

The calendar application allows users to add two basic kinds of scheduling objects: *events*, and *tasks*. Events and tasks are similar in that they both contain generic properties such as title, location, and details, but differ in that events specify a start and end time, whereas tasks specify a due date and indicate whether or not the task is complete.

Adding an Event

To add an event to the calendar, users select the Add event option from the day view page. The code for the day view page—which is stored as the `dayview.jsp` file—contains the following tag: `<util:link>`. This tag points to the `eventSess.jsp` file, which creates a new event. This new event—which at this point does not have any information entered by the user—is added to user sessions (see `<util:session>` in the *Portal Server Mobile Access Tag Library Reference* for more information about user sessions). From there, the tag `<util:forward>` passes control to the `editEvent.jsp` file and is displayed on screen as a series of individual prompts. Users are prompted for event information, such as Start (for start time), End (for end time), Title, Location, and Details. After filling in the required information, users see the following options in their browsers: Submit, Repeat, Invite, Remind, and Compose.

Submit

The Submit option submits the newly created event. When users select the Submit option, control flows to the `doUpdateEvent.jsp` file.This action brings up a success or failure confirmation message. Furthermore, this action brings up an option users can select to return to the day view page.

Repeat

The Repeat option allows users to specify the repeat frequency (daily, weekly, monthly, or yearly) for the event. Beyond the initial repeat frequency options, users have more specific options from which to select, as follows:

- For the daily repeat frequency, users can specify that the event repeat every day, on alternate days, or on a group of days, such as Monday, Wednesday, and Friday; Tuesday and Thursday; or Saturday and Sunday.
- For the weekly repeat frequency, users can specify that the event repeat every week, on alternate weeks, or once per week on a particular day.
- For the monthly repeat frequency, users can specify that the event repeat every month, on alternate months, or once per month on a specified date.
- For the yearly repeat frequency, users can specify that the event repeat every year, on alternate years, or once per year on a specified date.

After users select a repeat frequency and supply repeat information, they have the option of supplying one final parameter: the “Repeat until” value. As its name implies, this value sets the duration of time that the repeat should continue. The choices are to repeat until: Forever, Number of Instances, or Date.

The JSP files responsible for the repeat process begin with the prefix repeat, as in `repeatFreq.jsp`. The files involved are: `repeatFreq.jsp`, `repeatInt.jsp`, `repeatInterval.jsp`, `repeatIntOK.jsp`, `repeatOn.jsp`, `repeatOnOK.jsp`, `repeatOnMonthly.jsp`, `repeatOnWeekly.jsp`, `repeatOnYearly.jsp`, `repeatUntil.jsp`, `repeatUntilDate.jsp`, `repeatUntilDateOK.jsp`, `repeatUntilNumber.jsp`, and `repeatUntilNumOK.jsp`.

To see exactly which files link to which, consult the control flow reference charts for Calendar in the *Portal Server Mobile Access Developer’s Reference*.

Invite

The Invite option allows users to invite another person (the *invitee*) to the event. Selecting this option presents users with a number of methods for specifying the invitee, including: Quick Invite, Search, and Enter calID (calendar ID). When users select the Invite option, control flows from the `dayview.jsp` file to the `rsvp.jsp` file, to the `doRSVP.jsp` file, and then back to the `eventHm.jsp` file.

NOTE This feature is not supported when connecting to servers other than Sun Java System Calendar Server.

Remind

The Remind option enables users to set up email reminders. When users select this option, they are prompted for additional reminder data, (with one prompt per screen), as follows: the recipient's email address, the time (prior to the event) to send the email reminder, and a message to be included with the reminder. After users select the Remind option, control flows through the following files: `remindMail.jsp, remindTime.jsp, remindInterval.jsp, remindMsg.jsp, doRemind.jsp,` and then back to `eventHm.jsp`.

NOTE This feature is not supported when connecting to servers other than Sun Java System Calendar Server.

Compose

The Compose option allows users to modify the event being created. When users select the Compose option, control flows to the `editEvent.jsp` file.

Adding a Task

Users can also add tasks to the calendar. The method for adding a task to the calendar is similar to the method for adding an event. From the day view page, users simply select the Add Task option and proceed to fill in the requested information, such as the task title, location, due date, due time, details, and whether or not the task is already complete. When finished, users are given the following options: Submit, Remind, and Repeat.

Control flow through the various JSP pages for adding a task is similar to the flow for adding an event. First, control passes from the `dayview.jsp` file to the `taskSess.jsp` file, which creates a new task. This new task—which at this point does not have any information entered by the user—is added to user sessions. From there, the tag `<util:forward>` passes control to the `taskHm.jsp` file, which issues the task-related options provided when adding a task: Submit, Remind, and Repeat. The Remind and Repeat options take users through the same process as described in "Adding an Event" on page 15. The only difference between the process of adding an event and adding a task is that the source files responsible for adding a task have the suffix "Task," such as `remindMailTask.jsp`. While the source files for adding an event do not have this suffix, such as `remindMail.jsp`.

NOTE The task location, task remind, and task repeat features are not supported when connecting to servers other than Sun Java System Calendar Server.

Going to a Different Date

Users can reset the day view page to any arbitrary date by selecting the Go To option. When users select this option, the tag `<util:link>` transfers control from the `dayview.jsp` file to the `goto.jsp` file, which presents users with the following Go To options: “Today”, “Next week”, “Next month”, “Previous week”, “Previous month”, or a “Specific date”. As you might expect, when users select the “Next week” and “Next month” options, the calendar rolls forward by a week or a month; the “Previous week” and “Previous month” options roll the calendar back. The Today option rolls the calendar to the current date. When users select the “Specific date” option, control transfers from the `goto.jsp` file to the `date.jsp` file. Users can then type a specific date, which rolls the calendar to that date.

When the calendar is rolled forward or backward, control passes from the `dayview.jsp` file to the `doRoll.jsp` file, and then back to the `dayview.jsp` file, which displays the data for the new date. For convenience, the day view page also provides links to the next six days of the calendar, each of which link to the `doRoll.jsp` file as well.

Viewing an Existing Event or Task

As described in “The Day View Page” on page 15, the day view page summarizes all events and tasks for the day. Each summary is also an option that, when selected, displays the details of the particular event or task. For the purpose of this discussion, imagine a calendar containing one event (a party at 7:00 p.m.) and one task (buying refreshments, due by 5:00 p.m.).

To view the event or task, users select the desired option, which passes control from the `dayview.jsp` file to the `event.jsp` file or the `task.jsp` file.

To change event details, users select the Change option, which prompts users for event details (see “Adding an Event” on page 15). To delete the event, users select the Delete option, which leads to the `delete.jsp` file, and eventually back to the the `dayview.jsp` file.

As with the event details, changing or deleting the task is simply a matter of following the appropriate options.

The Mobile Address Book Application

The address book is a convenient application for storing personal contacts. The address book application works in conjunction with the mobile mail application to provide an efficient mechanism for sending mail to one or more recipients.

As with the calendar application, users launch the address book application from options provided on the mobile desktop. From the mobile desktop, the address book option allows users access to the address book channel. Users launch the address book application by selecting the address book option. This action passes control to the `list.jsp` file, which presents the address book home page to users. The address book home page summarizes the number of contacts currently stored and allows users to interact with the application.

Adding a Contact

To add a contact to the address book, users select the “Add contact” option, which uses the `<util:link>` tag to transfer control from the `list.jsp` file to the `add.jsp` file. The `add.jsp` file uses the `<util:include>` tag to link to GUI code in the `edit.jsp` file, which presents users with a series of individual prompts describing the new contact. The prompts include: “First name”, “Last name”, Phone, and Email. Control passes from the `add.jsp` file to the `doAdd.jsp` file (which submits the contact to the address book). The `<util:forward>` tag automatically returns control to the `list.jsp` file.

Adding a Group

The address book allows users to place multiple contacts into named groups. To create a new group, users select the “Add group” option, which passes control from the current page (the `list.jsp` file) to the add group page (the `addGroup.jsp` file). The add group page prompts users for the name of the group, then passes control to the members options page (the `addMembersOption.jsp` file), which presents the following menu: Search, Browse, and Enter Member.

Search

The Search option allows users to find a specific contact or group, or a list of contacts or groups, by entering search criteria and choosing the contacts or groups from the list of results. When users select this option, control passes from the `addMembersOption.jsp` file to the `searchMemberChoices.jsp` file, then to

doSearchMemberChoices.jsp file, and finally to the pickMemberChoices.jsp file. The pickMemberChoices.jsp file displays the results in the form of a choice list, from which users can select the contacts or groups to be added to the group.

Browse

The browse option allows users to scroll through the complete list of stored contacts and groups, without having to explicitly perform a search. When users select this option, control passes from the addMembersOption.jsp file to the browseMemberChoices.jsp file.

After fetching all of the contacts and groups, the browseMemberChoices.jsp file forwards control to the pickMemberChoices.jsp file. Again, this file displays the results in the form of a choice list, from which users can pick and select the contacts or groups to be added to the group.

Enter Member

The Enter Member option allows users to enter a contact or group directly if the exact name is already known.

Viewing, Changing, or Deleting a Contact

To view the details for a specific contact, users select the desired name from the list of contacts presented from the list.jsp file. If the selected option points to a single contact, control passes from the list.jsp file to the view.jsp file. If the selected option is for an entire group, control passes to the viewGroup.jsp file.

After selecting a single contact, users have three options: Compose (an email to the named contact), Change (the contact details), or Delete (the selected contact). Depending on which option users select, control passes from the current page to the compose.jsp file, change.jsp file, or the delete.jsp file.

Compose

This option is for composing an email message; for more information see “The Mobile Mail Application” on page 21.

Change

The Change option allows users to change the first name, last name, phone number, and email address of the selected contact. The JSP™ page responsible for generating this page (the `change.jsp` file) uses the `<util:include>` tag to include the GUI from the `edit.jsp` file. Existing properties appear with their current values, which users can change or accept. Control flow passes from the `change.jsp` file, which collects the information, to the `doChange.jsp` file, which submits the changes and returns users to the address book home.

Delete

To delete a contact from the address book, users simply select the Delete option. Control passes from the current page (The `view.jsp` file) to the delete page (The `delete.jsp` file) and then back to the address book home.

Viewing, Changing, or Deleting a Group

Users view, change, or delete a group from the address book following the same procedure as described for a single contact; however, in this scenario, control passes to group-specific JSP files such as the `changeGroup.jsp` file, `doChangeGroup.jsp` file, and the `deleteGroup.jsp` file.

When users view a group, they see a list of the group members and they have the option of composing an email to the entire group. Users can change a group by adding or deleting group members, or by changing the name of the group. Users delete a group in the same manner that they delete a contact, as explained in "Delete" on page 21.

The Mobile Mail Application

The mobile mail application is an email client that works in conjunction with the address book. Like the calendar and address book applications, mail is accessed through the main menu on the mobile desktop. The mail option provides access to the mail channel, which provides an option to launch the mail application.

Control passes to the mail home page, which is presented by the `summary.jsp` file. This display page summarizes the messages, and provides options for managing Inbox, composing a new message, and accessing other folders. This page can also provide the option for messages to be collected from a POP server if the Collections option has been properly configured. The possible options for the email home page appear as follows:

- Inbox
- Compose
- Folders
- Collections

Inbox

Inbox contains a From line at the top of the screen indicating the range of messages that are currently viewable. If Inbox contains more messages, an option labeled More appears at the bottom of the page. When users select that option, the current page is reloaded and the From line is updated to reflect the new range of messages that are then viewable. The range is updated on screen because, in the code, the start index value for the new range is set to increase by an increment of 1 over the end index value displayed in the previous range. This process loads the next set (page) of messages.

Views

Within Inbox or other mailboxes—though Inbox is the most common mailbox within which to set views—users see the messages that are visible depending upon which view, if any, Inbox is set to. If one or more views have been configured, the View option becomes available. When the View option is available, the option that by default is labeled MailHm changes to a menu option; therefore, the option no longer brings users to the mail home page, but instead offers users different view options. When users select the menu option, they see the “All messages” option and they see an option for each view that they have configured.

Views are pre-configured rules that determine which messages are visible on screen.

➤ **To Configure a View**

Users perform the following steps:

1. Log in to the standard Portal Desktop with an HTML browser
2. Scroll to the Mail communication channel
3. Select Edit Mail (the pencil icon)
4. Select the “Edit your mobile mail preferences” link
5. Under the Mobile Mail Views heading, configure the Views using the following options:

a. Rules Configuration

b. Views Configuration

c. Device Views

Users first configure a rule (Rules Configuration). Then they add the rule to a view (View Configuration). Finally, they apply the view to their specific mobile devices (Device Views). Once users have finished editing the mobile mail preferences page, they return to their mobile devices, navigate to the Views option, and select the view they want. The Views option appears only if users have already configured a view.

Email Messages

Each individual email message is summarized to show the sender's address and message subject. To view a message, users select the desired option, which passes control from the current page to the `getmsg.jsp` file, which fetches the message data from the server and forwards requests to the `message.jsp` file, which displays the subject and body of the message as well as the applicable options.

The possible options are the "List attachment(s)", Next, and Previous options. When available, these options appear at the bottom of the message body. The Next option, Previous option, or both (Next and Previous options) appear when next and previous messages are available to view. When users select one of these options, the respective message (next or previous) is loaded. The "List attachment(s)" option appears when one or more attachments are attached to the message. When users select this option, control is passed to the `attach.jsp` file, and users are presented with a list of available attachments.

After users have opened a message, they have a menu of other options available to them. This menu option is linked to the `menu.jsp` file. When users select this menu option, the menu page appears on their browsers. This page provides users with the following options: Reply, Delete, Forward, Move To, Copy To, Add to Addr, "View header", Mailbox.

Reply

The Reply option provides users with a dialog for replying to the sender's message. When users select this option control passes to the `reply.jsp` file, which asks users if the reply should go only to the sender, or to everyone included in the original email. To answer this question, users select To Sender or To All, both of which pass control to the `compose.jsp` file. This action brings up the compose page on users' browsers, where they can compose a message (see "Compose" on page 25).

Delete

The Delete option simply deletes the message. Control passes from the current page (the `menu.jsp` file) to the `delete.jsp` file, and then back to Inbox.

Forward

The forward option forwards the message to a specified recipient, which passes control to the `pickaddr.jsp` file. Users are then able to select a recipient from the list of stored contacts. Users select the recipients to include, which passes control to the `compose.jsp` file. If no address book channel is configured, control passes to the `compose.jsp` file directly. For details about the compose page, see "Compose" on page 25.

Move To

The Move To option moves the message to a specified folder, deleting the message from Inbox in the process. When users select this option, control passes to the `getfolders.jsp` file, which fetches the folder names from the server, then forwards users to the `folders.jsp` file, which displays the available list of folders. A New Folder option also appears, enabling users to create a new folder if desired (see "Folders" on page 25).

Copy To

The Copy To option behaves exactly as Move To, except that the original message is left in Inbox after the operation.

Add to Addr

The Add to Addr option adds a contact to the address book. When users select this option, control passes from the current page (the `menu.jsp` file), to the `addpab.jsp` file (short for add to personal address book), which prompts users—one screen at a time—for the following contact information: first name, last name, phone, and email address. For the email address, the sender's address is automatically provided, but the sender's address can be edited by users if desired. After users supply the contact information, a Mailbox option appears which returns users to the current mailbox folder, Inbox in this case.

View Header

The View header option links to the `header.jsp` file, and displays the following email header fields: from, date, subject, and to. This screen simply displays information. Users are not presented with additional choices.

Mailbox

The Mailbox option returns users to the currently selected mailbox. In this case, when users select this option, control returns to Inbox.

Compose

The compose page (the `compose.jsp` file), is where users compose a new email message. The Compose option is available in the mail home page, which is presented by the `summary.jsp` file:

When users select this option, control is first passed to the `pickaddr.jsp` file, (even when an address book communication channel has not been configured), then control is passed to the `compose.jsp` file. The compose page prompts users for standard email fields, such as: to, cc, subject, and message body. The interface is intuitive; users simply supply the requested fields, submit the message, and return to Inbox.

An email message composed on a mobile device can be time-consuming; therefore, users are allowed to define up to nine preset messages. These predefined replies can be selected for inclusion at the beginning of the compose screen.

➤ **To Create a Predefined Reply**

Users perform the following steps:

1. Log in to the standard Portal Server with an HTML browser
2. Scroll to the Mail channel
3. Click Edit Mail (the pencil icon)
4. Click the “Edit your mobile mail preferences” link.

   Predefined replies have a 30-character maximum.

Folders

After users click the Folders option in the mail home page— which is presented by the `summary.jsp` file—they are able to view or create a specific folder. When users select the Folders option, control is passed to the `getfolders.jsp` file, which fetches the folder names from the server and forwards requests to the `folders.jsp` file, which then displays the available list of folders. A New Folder option also appears, allowing users to create a new folder if needed.

When users select the New folder option, control passes from the current page (the `folders.jsp` file), to the `newFd.jsp` file, which prompts users for the name of the folder. Control then passes to the `doNewFd.jsp` file, which creates the new folder. Users then return to the folders page by following the Folders option that appears after the new folder has been created.

When users select an existing folder, control passes to the `folderMenu.jsp` file, which provides users with three options from which to select: Open, Rename, or Delete.

Open

The Open option reloads Inbox screen (passing control to the `doInbox.jsp` file), substituting the selected folder data for Inbox data.

Rename

The Rename option passes control to the `renameFd.jsp` file, which prompts users to enter a new name for the folder.

Delete

The Delete option passes control to the `delFd.jsp` file, which deletes the folder.

All three links eventually present users with a Folders option that returns control to the `folders.jsp` file.

Collections

The Collections option can be made available in the mail home page, which is presented by the `summary.jsp` file. The Collections option allows users to view mail on their mobile devices that was collected from a POP mail server.

➤ **To Configure a POP Mail Server for Mail Retrieval:**

Users perform the following steps:

1. Log in to the standard Portal Desktop with an HTML browser.
2. Scroll to the Mail channel.
3. Click Edit Mail (the pencil icon).
4. Click the “Edit your mobile mail preferences” link.
5. Under the Mobile Mail POP Collections heading, select Add.
6. Populate the fields of the POP Information box with the details for the POP server to which users want to connect.
7. Click Finished.

Once users have finished editing the mobile mail preferences page, they return to their mobile devices where they see the Collections option in the mail home page.

Chapter 2

Developing Voice Applications

This chapter provides information about Sun Java System Portal Server Mobile Access voice application development. The chapter contains the following sections:

- Understanding Voice Applications
- File System Directories for Dialogs, Grammars, and Prompts
- Porting the Voice Environment and Applications
- Localizing Voice Applications

Understanding Voice Applications

Voice applications are developed similarly to other Sun Java System Portal Server applications. The best approach for developing voice applications is to develop them in VoiceXML and then to integrate them with Portal Server software by developing a custom provider. The Notes and Personal Notes voice applications both use this approach.

This section discusses the following topics:

- Voice Application Prerequisites
- A Voice Application Example
- Building a Voice Application

Voice Application Prerequisites

The pre-requisites for integrating new voice applications with Portal Server Mobile Access software are:

- Voice components of Mobile Access software are certified against the Nuance Voice Platform, which includes a VoiceXML 2.0-compliant voice browser. Because of differences between VoiceXML interpreters, the voice components might need to be ported to run on other platforms (see to "Porting the Voice Environment and Applications" on page 37 for further information). Portal Server software and voice applications should use the same VoiceXML browser.
- Mobile Access software supports VoiceXML applications only. If you have a legacy voice application that was not built using VoiceXML, you must provide a VoiceXML wrapper for that legacy voice application. This wrapper will integrate with Portal Server software and manage the voice session with the non-VoiceXML application.
- If you are not familiar with building voice applications using VoiceXML, consult a book on VoiceXML programming. The W3C VoiceXML 2.0 specification is the definitive reference on the programmatic elements of the language.

A Voice Application Example

The best way to start building voice applications is to examine the Notes application included with Portal Server software. The Notes application consists of a provider (`NotesProvider`) that uses template files for each type of access device (web browser, wireless device, and voice browser). The voice application template files for the `NotesProvider` are stored in the following directory:

```
/etc/opt/SUNWps/desktop/default/NotesProvider/vxml/Nuance
```

These files contain VoiceXML code in addition to template tags. The tags provide dynamic content when the dialogs are accessed. For example, the `content.template` file uses `[tag:count]` to retrieve the number of notes and `[tag:note]` tag to speak the notes using text-to-speech:

```
<prompt bargein="true"> [tag:note]</prompt>
```

The prompts for this voice application are Microsoft Windows audio (`.wav`) files, stored in the following directory:

```
/opt/SUNWwbsvr/docs/voice/en_US/prompts/gary
```

This path is constructed at runtime by concatenating the root directory (`/voice`), the locale identifier (`en_US`), a prompts sub-directory (`prompts`) and a persona (`gary`). The resulting path `/voice/en_US/prompts/gary` is relative to the web container directory `/opt/SUNWwbsvr/docs`.

Finally, each voice application must provide a grammar that allows the application to be selected from the voice desktop channel chooser. The grammar for the Notes application (`notes.grammar`) is located in the following directory:

`/opt/SUNWps/web-src/jsp/default/Notes/vxml/Nuance/grammars`

This directory contains the following Nuance GSL grammar expression:

```
Notes [ (notes ?channel) ]
```

This allows users to select the channel by speaking the phrase `notes` or `notes channel`.

Building a Voice Application

To create a new voice application, you need to build a Portal Server custom provider, or extend an existing provider. For details on creating a custom provider, see the information on leaf providers in the *Portal Server Developer's Guide.*

Most interactive voice applications use dynamically-generated content. For example, a weather application might report the weather for a particular region when you speak the name of a city or postal code. The dynamic content (the weather in this case) is retrieved from a weather service at runtime. For this reason, voice application dialogs are typically generated dynamically using techniques such as JavaServer™ Pages (JSP™) technology.

Providers *could* generate the VoiceXML dialogs programmatically, but the simplest approach is to build template files that contain the static dialog code, and use tags that are interpreted at runtime to retrieve the dynamic content.

The following steps describe how to build a provider that implements a voice-enabled weather application:

1. Develop a dialog design.

   Most voice applications consist of a set of dialogs. Each dialog is responsible for one part of user interaction.

   You can use a flow chart to represent the design of your voice application. The flow chart should include phrases spoken by users, shown as transitions between the dialogs. Or, you can develop a script where the conversational flow between the voice application and users is listed in chronological order.

   Either way, you must handle cases where users say something that was not understood, or the voice application did not receive any input.

2. Build a prototype of your voice application in VoiceXML.

   For dynamic content, begin by simply including static text as a placeholder content. For example, in a weather application, you might always speak the same report using a `<prompt>` statement:

   ```
   <prompt>
   Here's the weather forecast for Santa Clara, California.Today
   will be mostly sunny, with a high of 75 and a low of 68
   degrees Fahrenheit.
   </prompt>
   ```

   The static content will be replaced with dynamic content once the prototype is complete.

3. Test your application, exploring all possible dialog interactions.

4. Integrate the application with the Portal Server software.

   Adding a voice application to the Portal Server software involves building a custom provider. The simplest approach is to use template files like the NotesProvider described in the previous section. The template approach allows you to take the VoiceXML dialogs from the prototype and use them directly with your provider.

5. Identify static placeholder text in your VoiceXML prototype.

   Review your VoiceXML prototype and identify the places where you use static content as placeholders for dynamic content. In each case, you must build a custom tag that can generate the appropriate dynamic content at runtime. For example, the weather report might be implemented using a custom `weather` tag.

6. Build a custom provider using templates.

   See information on leaf providers in the *Portal Server Developer's Guide* for details on building a custom provider that uses templates. Follow the instructions for creating a new custom provider. Implement support for the custom tags required for dynamic content in the voice application.

7. Edit the VoiceXML files.

   While building the custom provider, you must make some changes to the VoiceXML files:

   - Replace the static content with tags. For example, the static weather report in "Build a prototype of your voice application in VoiceXML." on page 32 would be replaced with the following, assuming you have implemented support for a `weather` tag in your provider:

     ```
     <prompt>
     [tag:weather]
     </prompt>
     ```

   - Change the file extensions of the VoiceXML files from `.vxml` to `.template`. You must modify references to other dialogs in the VoiceXML code.

8. Move all files to the appropriate directories.

   The files comprising your voice application must reside in specific directories. For a discussion of these directories, see "File System Directories for Dialogs, Grammars, and Prompts" on page 35.

9. Complete the installation of the new provider.

   The basic steps are as follows. For detailed instructions, see information on implementing a custom provider in the *Portal Server Developer's Guide*.

   - Compile and install the provider class file in the correct location.
   - Install any resource bundle files.
   - Create an XML channel entry for the provider and update the Display Profile.
   - Add the channel to the `VoiceJSPDesktopContainer`.
   - Add the channel to the `Available` and `Visible` list so that users can select the channel.

10. Test the new provider.

Create user accounts with a numeric user name and a numeric PIN, then call into the Portal Server software and enter that information when required.

At the main menu, speak the `add a channel` command to add the new channel from the list of available channels:

User:

Add a channel.

System:

Sure;

Here's the list of channels you can add:

E-Mail, Calendar, Weather

That's it.

Tell me which one you want to add, or say cancel.

User:

Weather

System:

All right;

Weather has been added.

Would you like to go there?

User:

Yes

System:

OK, Weather.

When you're done, say *main menu*.

Here's the weather forecast for Santa Clara, California:

Today will be mostly sunny, with a high of 75 and a low of 68 degrees Fahrenheit.

OK, we're back at the portal main menu. What’s next?

File System Directories for Dialogs, Grammars, and Prompts

Portal Server software has specific directories for the various provider components. This section discusses the directories for:

- Dialogs
- Grammars
- Prompts

Dialogs

The dialogs for each voice application are stored in a separate directory that includes the name of the application, the presentation format (vxml) and the name of the voice browser vendor (Nuance).

For a weather application, the dialog files would be stored in:

`/etc/opt/SUNWps/desktop/default/`*`weather`*`/vxml/Nuance`

where *weather* is the name of the new application.

Grammars

If your application uses external grammar files, they should be stored in the web server's document root, or in some other well-known location within the Portal Server web application.

To make the application accessible from the voice Portal Desktop, you must create a second grammar file that allows users to select the application. The grammar for the channel must be unique across all of the voice-enabled channels. For consistency, the grammar should allow users to optionally speak the word `channel` after the name of the channel.

For example, the following grammar allows `weather` or `weather channel`:

```
Weather [ (weather ?channel) ]
```

Name this file `weather.grammar` and store the file in the following directory:

`/opt/SUNWps/web-src/jsp/default/weather/vxml/Nuance/grammars`

Prompts

The voice prompts are located in the following directory:

`/opt/SUNWwbsvr/docs/voice/en_US/prompts/gary`

The path element `gary` is the name of the default persona—the person whose voice appears on the recording. If you record new prompts, you should create a new directory for the new persona. This new directory could be named after the person who recorded the prompts.

For example:

`/opt/SUNWwbsvr/docs/voice/en_US/prompts/cheryl`

Voice prompt file names use the following naming convention:

- Use the words of the prompt in lower case as the file name.
- Use an underscore character between words of the phrase. For example: `word_word.`
- Truncate the file name at 50 characters, leaving an additional 4 characters for the `.wav` extension

  To use this prompt directory in your voice application, prepend the path `/voice/en_US/prompts/`*`cheryl`* (or the path to your prompts) before the prompt file name.

  For example, if your prompts were stored in a `prompts/` sub-directory, replace this statement:

  ```
  <audio src="'thats_it.wav'" />
  ```

  with:

  ```
  <audio src="'/voice/en_US/prompts/cheryl/thats_it.wav'" />
  ```

  For localization convenience, you might want to define some VoiceXML variables in your application's root document, and construct the path from this:

  ```
  <var name="root" expr="'/voice'" />
  <var name="locale" expr="'en_US'" />
  <var name="prompts_directory" expr="'prompts'" />
  <var name="persona" expr="'cheryl'" />
  <var name="promptPath" expr="root + '/' + locale + '/' +
      prompts_directory + '/' + persona + '/'" />
  ```

Then use the *promptPath* variable as follows, using `expr=` instead of `src=`:

```
<audio expr="promptPath + 'thats_it.wav'" />
```

Porting the Voice Environment and Applications

To provide voice functionality with non-Nuance VoiceXML platforms, the Portal Server voice environment and voice applications must be ported.

The following issues are commonly encountered while porting:

- Each VoiceXML dialog file contains an XML DTD header, and the XML DTD header used by your voice application must be correct. To determine the correct XML DTD header, see the documentation for the voice browser you are using.
- Voice browser vendors sometimes use different grammar file formats. These could be proprietary, or based on evolving industry standards. Check to see if the grammar used by your voice browser is compatible with the Nuance format used by the Portal Server software, and modify the grammars if necessary. Some grammars might be inline in VoiceXML dialogs, or in external grammar files.
- If you develop VoiceXML applications that use client-side scripting, such as ECMAScript or JavaScript, be aware of interpreter differences between voice platforms.
- Some voice browsers might not support the full set of VoiceXML tags. The voice browser documentation usually lists which tags the browser supports. You might need to modify VoiceXML code to remove unsupported tags.
- Some VoiceXML tags behave differently between voice browsers. Although the code might execute, the behavior could change subtly. Thoroughly test all dialog states in your application, even if the dialog states appear to execute correctly.
- If pre-recorded prompts are not played correctly, you might need to change the encoding from the default 8-bit, 8kHz, mulaw sphere encoded WAV audio file format.

Localizing Voice Applications

Voice applications are more locale-dependent than conventional software applications. Not only must the language that the computer uses to communicate with users be modified for user locales, but the voice interface must also be modified so that interface recognizes the language spoken by users. There might even be differences within the same language based on location. Localizing a voice application therefore requires careful attention to language and regional differences.

Localization of a voice application involves:

- Re-recording Voice Prompts
- Grammar Translation
- Modifying Pre-Recorded Prompts to Match Grammar Changes
- Updating Concatenated Phrases
- Translating Text-To-Speech Prompts

Re-recording Voice Prompts

Mobile Access software uses a naming scheme for recorded prompts, where prompt file names are based on the words in the prompt with underscores between words.

For example, the prompt:

`Here are your notes`

is named

`here_are_your_notes.wav`

For long phrases, the file name is truncated at 54 characters with 50 for the file name and 4 for the extension `.wav`.

The phrase:

`Tell me which channel you want to add, or say` *`cancel`*

would be named:

`tell_me_which_channel_you_want_to_add_or_say_cance.wav.`

Using English phrases for prompt names greatly improves the readability of VoiceXML code for English speaking developers.

To localize an application that contains prompts, you do not need to change the prompt file names. Instead, the prompts can be re-recorded in the new language and saved with the same file name.

The prompt:

`Here are your notes`

would become:

`Voici vos notes`

in French, but the file name would remain `here_are_your_notes.wav`. This approach allows the language used by the application to be changed without editing any of the VoiceXML `<audio>` tags.

Grammar Translation

VoiceXML applications use grammars to define the phrases that users can speak in any given dialog. For example, when logging into the Portal Server software, users are prompted to enter an account number. In this case, the grammar allows users to speak a sequence of numbers. Once logged in, users can access a channel by speaking the channel name, such as *email*.

In VoiceXML applications, grammars can be included *inline* in the VoiceXML dialog code, for example:

```
<link next="#goodBye">
    <grammar scope="dialog">
        [
            ( goodbye )
            ( exit )
            ( quit )
         ]
    </grammar
</link>
```

The other option is to store the grammars in a file and reference that file form the VoiceXML dialog.

For example:

```
<form id="channelcommand">
    <field name="action" slot="action">
    <grammar src="grammars/overview.grammar#NextAction"/>
```

In this case the grammar is located in the file `grammars/overview.grammar`.

Voice applications, such as Personal Notes and Bulletin Board, generally use inline grammars. However, external grammar files can also be used as well.

For external grammar files, localization involves replacing the English words or phrases in the grammar file with their equivalents in the new language. Do not simply replace words with their dictionary equivalents. Instead, choose words or phrases that native language speakers would use in every day conversation.

When several ways of saying the same thing are available, include these alternatives in the grammar. For example, in the previous example, users can exit by saying `goodbye`, `exit`, or `quit`.

For inline grammars, you must identify which files contain inline grammars. Search for files that contain the string `<grammar>` but do not include a `src=` parameter (which indicates an external grammar file). Replace the words or phrases as you would in the case of external grammar file, but be careful not to inadvertently modify other parts of the VoiceXML code.

Modifying Pre-Recorded Prompts to Match Grammar Changes

Some voice prompts contain instructions on how to interact with the system, such as `to end your session say` *goodbye*. In this case, the application grammar is defined to recognize the word *goodbye*. When localizing a voice application, you must take care to ensure that when you change a grammar, you also modify any prompts that refer to that grammar. Typically, you will find audio prompts that give instructions to users in `<noinput>`, `<nomatch>`, and `<help>` VoiceXML tags.

Before the recording artist re-records these prompts for the new language, make a note of any grammar changes, and update the localized prompt phrase to match the grammar.

Updating Concatenated Phrases

Sentences in voice applications are frequently constructed from individual phrases and words. For example, the phrase `Today is Tuesday April 8th, 2003` might be constructed from the following eight words or phrases: `Today is`, `Tuesday`, `April`, `eighth`, `two`, `thousand`, and `three`. The VoiceXML code plays these prompts in order.

Localizing the application might require that words be concatenated in a different order. Recording the prompts in the localized language might not result in a correctly structured sentence.

This problem has two solutions:

1. Test the localized application by interacting with the application to detect any instances where the phraseology is incorrect. This is the simplest approach for localization teams who are not familiar with VoiceXML.
2. Perform a code review, identifying prompt concatenation in the code, and making changes to the prompts as necessary. In some cases you might need to add new prompts to account for significant changes in sentence structure. You might need to go back to the recording studio to record new prompts.

Concatenated phrases might also suffer from cadence issues. Cadence is the way that individual words and phrases flow within a sentence. In some languages, words flow together without pauses. This could require the removal of silence at the beginning or end of recorded prompts, or in some cases, the recording of a single phrase to replace several concatenated words.

Cadence issues are usually discovered during testing and can often be resolved with careful prompt editing. If you edit or re-record a prompt to work well in one concatenation, the prompt might not work correctly if used elsewhere in a different part of the sentence. If you make a change to a prompt in one dialog, check all other cases where that prompt is used to ensure that the change does not adversely affect them.

Sometimes the pronunciation of a word changes depending on the immediately preceding or following words, or if the language has masculine and feminine forms of words, depending on the gender of the object. Review the prompt phrases before recording and make notes to the recording artist where a particular pronunciation is required. If the article in the sentence is only known at run-time, you might need to add VoiceXML code to select the correct pronunciation depending on the gender of the article.

Translating Text-To-Speech Prompts

In addition to pre-recorded prompts, some voice applications use text-to-speech (TTS) prompts. These prompts appear as English text in VoiceXML code within `<prompt>` statements. For example:

```
<prompt>Please say yes or no</prompt>
```

TTS prompts can also be used in conjunction with pre-recorded prompts:

```
<prompt><audio src="you_have.wav"/> 5 <audio src="unread_messages.wav"/>
</prompt>
```

In this example, TTS is used for the word `five` in the phrase `you have five unread messages`.

Finally, VoiceXML variables can be used in TTS prompts:

```
<prompt><value expr="num_messages"/></prompt>
```

In this example, the digit 5 and the variable `num_messages` are spoken using TTS. No localization work is required because the TTS engine for the new locale automatically speaks the number in the new language. However, variables can also be assigned values that correspond to English words or phrases that the TTS engine will not translate. In such cases you must identify any place in the VoiceXML code where English language strings are assigned to variables. Look for `<assign>` tags such as:

```
<assign name="prompt" expr="'OK, got it!'"/>
```

You must change any embedded English language words that would be spoken using TTS. The easiest way to identify these prompts is to search for `<prompt>` tags.

Chapter 3

Using the Mobile Access Public APIs

This chapter provides information about the public APIs exposed by the Mobile Access software. The chapter contains the following sections:

Overview

This release exposes three public APIs that you can access while developing your own mobile applications: the Tag Library API, the Desktop API, and the Rendered Desktop API. The Javadoc for the Desktop API and the Rendered Desktop API can be found on the system where Portal Server is installed, which is at this link:

`http://`*`hostname:port`*`/portal/javadocs/ma/index.html`

In addition, the Tag Library API is described in the following section, which lists all of the publicly exposed classes that are provided in this release.

The Tag Library API

The Tag Library API provides a mechanism for tracking user state across multiple requests. This API also enables developers to change the behavior of the current tag library by creating their own custom subclasses.

The Tag Library API consists of the following base and application-specific classes:

- `Context`, `ContextCache`, and `ContextTag`
- `ABContext`, `ABContextCache`, and `ABContextTag`
- `CalContext`, `CalContextCache`, and `CalContextTag`
- `MailContext`, `MailContextCache`, and `MailContextTag`
- `UtilContext`, `UtilContextCache`, and `UtilContextTag`

The base classes, `Context`, `ContextCache`, and `ContextTag`, are the primary focus of this chapter.

The Desktop API

The Desktop API provides methods for accessing and manipulating rows and channels. The API contains the following classes:

- `WirelessContainerProvider`
- `WirelessJSPContainerProviderAdapter`
- `JSPNativeContainerProvider`

Usage instructions for the preceding classes are available from the Javadoc on the system where Portal Server is installed, which is at this link:

`http://`*hostname:port*`/portal/javadocs/ma/index.html`

The Rendered Desktop API

The Rendered Desktop API provides classes related to the rendering and delivery of AML markup. The API contains the following classes:

- `RenderingContainerProvider`
- `JSPRenderingProvider`
- `RenderingUtil`

As with the Desktop API, usage instructions for the Rendered Desktop API are also available on the system where Portal Server is installed, which is at this link:

`http://`*hostname:port*`/portal/javadocs/ma/index.html`

Understanding the Context Class

The `Context` class resides in the `com.sun.portal.wireless.taglibs.base` package, and defines the core state-tracking functionality that your custom subclasses will inherit.

To obtain an instance of this class, invoke its `getContext` method:

```
public static Context getContext(PageContext pageContext, String
contextClassName, String contextCacheClassName, String contextType);
```

Like all `public static` methods, `getContext` should be invoked by class name, not by object reference. When invoked, this method will first attempt to retrieve the context from the `pageContext` and `contextCache` objects. If both attempts fail, the method will instantiate a new context, placing a copy into the cache before returning. This method is most commonly invoked by `Context` subclasses as part of their own `getContext` implementations.

When the `getContext` method creates a new `Context` object, the method triggers an initialization phase that sets values for many of its properties. The process is initiated by invoking either of the following methods:

- `public void init(HttpServletRequest request, SSOToken session, String serviceName, SSOAdapter ssoAdapter) throws Exception`
- `public void init(HttpServletRequest request, SSOToken session, SSOAdapter ssoAdapter) throws Exception`

The first version initializes the context and associates the context with an Identity Server service. This version is typically invoked with `super.init(...)` from within the `init(...)` implementations of specific `Context` subclasses.

The second version simply initializes the context. This version is typically invoked by the context framework, such as `Context.getContext(...)`, immediately after instantiating a new `Context` object.

Regardless of which version is invoked, the following tasks are always performed during context initialization:

- The `session` and `ssoAdapter` parameters are assigned to protected instance variables of the same name. Both objects will become directly accessible in your subclass, as well as through the public methods `getSession()` and `getSSOAdapter()`.
- The ID value of user sessions becomes accessible through the public method `getSessionID()`, and the configuration name of the `ssoAdapter` becomes available through the public method `getConfigName()`.
- The context instance is registered as an `SSOTokenListener` on user `sessions`. `SSOTokenEvent` notifications will be received by this instance through its `public void ssoTokenChanged(SSOTokenEvent evt)` method. All subclasses will inherit this behavior, and as such, will automatically receive `SSOTokenEvent` change notifications for events such as logouts and timeouts. Your subclass might need to override the `ssoTokenChanged` method to provide additional behavior as necessary.
- User client types are detected and made accessible through the `public String getClientType()` method. Once established, user content types become accessible through the `public String getContentType()` method.
- User locales are determined and made accessible through the methods `public Locale getLocale()`, `public Locale getUserLocale()`, and `public String getUserLocaleString()`.
- User time zones are retrieved and stored in the `protected TimeZone` time zone instance variable.

The remaining methods of this class are simple methods for reading and writing the context's properties. For example, `public Context getParentContext()` and `public void setParentContext(Context)`, which get and set the parent context. Such methods are fully documented and available on the system where Portal Server is installed, which is at this link:

`http://`*hostname:port*`/portal/javadocs/ma/index.html`

Understanding the ContextCache Class

The `ContextCache` class, as its name implies, is a simple cache mechanism for storing multiple `Context` instances. This class exists for essentially one reason: to give a unique name to the cache into which a particular type of context will be stored.

The `ContextCache` class resides in the `com.sun.portal.wireless.taglibs.base` package.

Typically, for every major `Context` subclass (for example, `MailContext`, `CalContext`, and `ABContext`), there exists a corresponding `ContextCache` subclass (for example, `MailContextCache`, `CalContextCache`, and `ABContextCache`).

To obtain an instance of this class, invoke the following method, passing in the name of the context to be retrieved:

```
public static synchronized ContextCache getInstance(String className)
```

To add / remove contexts to or from the cache, invoke `put` or `get`:

- `public void put(Context context, String configName)`
- `public synchronized void put(Context context, String configName, boolean listener)`
- `public synchronized Context get(SSOToken session, String configName)`

Understanding the ContextTag Class

The `ContextTag` class resides in the same package as `Context` and `ContextCache`. This class is a simple class that represents a *context* tag.

A context tag represents the session state for an valid authenticated connection to a backend service. This tag validates that the session is still valid and provides a context for other tags to make requests to that service. At the first request, a connection is made to the backend service using configuration and authentication information. The session service context or state is stored in the session so that the context or state is available across requests.

A context tag does the following:

- The service context is retrieved from the session.
- If there is no context, a new service context is created and stored into the session.
- The service configuration (for example, host, port) and user authentication (for example, credentials) are used to connect to the service.
- If a valid session is created, the service context is stored into the session

The `ContextTag` class defines an abstract `findContext` method, which finds the context this tag represents:

```
public abstract Context findContext() throws Exception;
```

Subclasses must implement this method to return the appropriate context for the session. For an example of this, see "Extending the Tag Library" on page 49.

Creating Custom Subclasses

This section presents guidelines that you should follow when deriving your own classes from `Context` or `ContextCache`.

- If you are creating a direct `Context` subclass, you should define the following `String` constants, since their values will be required when obtaining a context: `CONTEXT_CLASS_NAME`, `CONTEXT_CACHE_CLASS_NAME`, and `CONTEXT_TYPE`. The first two constants should contain the fully-qualified class names for your `Context` and `ContextCache` subclasses. The last constant declares the name of the context type appropriate to this new `Context` subclass.
- If you are creating a direct `Context` subclass, you should define a unique value for `CONTEXT_TYPE` that is descriptive of the type of service provided by the new context. If you are simply extending an existing major `Context` subclass (for example, `MailContext`, `CalContext`, or `ABContext`), you should just inherit the value for `CONTEXT_TYPE` that is defined in those subclasses.
- If you are creating a direct `Context` subclass, you should define a `getContext` method for obtaining its instances. The body of the method can simply call `Context.getContext`, casting the result to the current class:

```
public static <YOUR_SUBCLASS_NAME> getContext(PageContext
pageContext)throws Exception
{
return (<YOUR_SUBCLASS_NAME>) Context.getContext(pageContext,
CONTEXT_CLASS_NAME,CONTEXT_CACHE_CLASS_NAME,CONTEXT_TYPE);
}
```

- If you are extending an existing `Context` subclass, such as `ABContext`, the result can be cast to that class instead:

```
return (ABContext) Context.getContext(pageContext,
CONTEXT_CLASS_NAME,CONTEXT_CACHE_CLASS_NAME,CONTEXT_TYPE);
```

- When overriding either version of `init`, make sure that your subclass invokes `super` as its first line:

```
// Inside a Context subclass
public void init(HttpServletRequest request, SSOToken session,
SSOAdapter ssoAdapter) throws Exception{

super.init(request,session,<your_custom_service_name>,ssoAdapter);


... // your app-specific code from here on
```

  Failure to do so will prevent the context from being initialized correctly.

- The base `Context` class registers itself as a listener for changes fired from user sessions, and your subclass will inherit this base behavior. You might want to override this method to perform additional tasks beyond what is defined in the parent implementation. Typically this will involve code for cleaning up the context when the session becomes invalid.
- If you have created a direct `Context` subclass, you should also create a direct `ContextCache` subclass that corresponds to the direct `Context` subclass. For example, `IMContext` and `IMContextCache`. Using this example, the new cache class need only contain the following:

```
package com.whatever.im;

import com.sun.portal.wireless.taglibs.base.*;

public class IMContextCache extends ContextCache { }
```

- If you are extending an existing major `Context` subclass, you should not create a new `ContextCache` subclass. Instead, you should use the same `ContextCache` subclass that is already defined.

Extending the Tag Library

This section provides an example of how to extend the current Tag Library by defining new `Context`, `ContextCache`, and `ContextTag` subclasses.

1. Extend the `Context` class.

```
public class NewCalContext extends CalContext {

  public static String CONTEXT_CLASS_NAME=
  “com.sun.portal.wireless.taglibs.cal.NewCalContext”;

  public static final String CONTEXT_CACHE_CLASS_NAME=
  “com.sun.portal.wireless.taglibs.cal.NewCalContextCache”;

  public static final String CONTEXT_TYPE = “Calendar”;

  protected boolean newProp = false;

  public void setNewProp(boolean newProp){
    Util.logMessage(“NewCalContext.setNewProp():” +
    “Setting New to” + newProp + “, instance =”+ this);
    this.newProp = newProp;
  }

  public boolean isNewProp(){
    Util.logMessage(“NewCalContext.isNewProp():Retrieving
New=”
    + newProp + “, instance =” + this); return newProp;
  }

  public static CalContext getContext(PageContext
pageContext)
     throws Exception {
    return (NewCalContext) Context.getContext(pageContext,
   CONTEXT_CLASS_NAME,CONTEXT_CACHE_CLASS_NAME,CONTEXT_TYPE);
  }
```

2. Optionally, extend the `ContextCache` class.

   If your application does not need access to the original context object, extending the `ContextCache` class is not necessary.

   However, if your application needs access to both the new and original context, you must create a new `ContextCache` subclass. You do not need to override any methods.

   ```
   public class NewCalContextCache extends CalContextCache { }
   ```

3. Write a `ContextTag` class that refers to the new `Context` subclass.

```
public class NewCalContextTag extends CalContextTag {
  public NewCalContextTag()
    super();
  }
  public Context findContext() throws Exception {
    String configNameKey = NewCalContext.CONTEXT_TYPE +
                           "configName";
    String ssoAdapterKey = NewCalContext.CONTEXT_TYPE +
                            "ssoAdapter";
     computeConfigName(pageContext, configNameKey, ssoAdapterKey,‘
                      CalContext.SSO_CONFIG_TYPE);
    return NewCalContext.getContext(pageContext);
  }
}
```

Use this new `ContextTag` in the tld to refer to the Tag class.

```
<name>context</name>
  <tagclass>
  com.sun.portal.wireless.taglibs.cal.new.NewCalContextTag
  </tagclass>
  <bodycontent>JSP</bodycontent>
  <info> New Calendar context tag </info>
  <attribute>
    <name>config</name>
    <required>false</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
</tag>
```

Global Namespace Request Parameters

Global namespace requests parameters are provided in this release for the rendered and native Portal Desktop and for the voice Portal Desktop. This section lists them.

Rendered and Native Desktop

- `content`
- `m`

- moveUp
- moveDown
- nextURL

Voice Desktop

- action
- c
- container
- last
- mode
- nextURL
- provider

Chapter 4

Deploying Online Help for the Mobile Portal Desktop

Although Sun Java System Portal Server Mobile Access software does not provide online help for the mobile Portal Desktop, you can create your own online help files for existing sample channels or for new channels you create for the mobile Portal Desktop.

This chapter outlines the process for deploying online help files to the mobile Portal Desktop. The topics discussed include:

- Adding Help Options
- Updating the Channel Display Profile
- Creating a Help File
- Installing Help Files in the File System
- Deploying Help Files

Adding Help Options

You should add a Help option (link) to the channel content. How you do this depends on how the channel is created—by rendered Portal Desktop JavaServer Pages™ (JSP™) technology, by native Portal Desktop JavaServer Pages technology, or by native Portal Desktop templates.

The topics in this section include these methods:

- Rendered Portal Desktop JSPs
- Native Portal Desktop JSPs
- Native Portal Desktop Templates

Rendered Portal Desktop JSPs

Use `getHelp` from the desktop tag library to add a help option to the mobile channel's content. For instance:

```
<%@ taglib uri="/tld/desktop.tld" prefix="dt" %>
....
<AmlControlMenu url="<dt:getHelp/>" label="Help"/>
```

Native Portal Desktop JSPs

Use `getHelp` from the desktop tag library to add a help option to the mobile channel's content. For instance:

```
<%@ taglib uri="/tld/desktop.tld" prefix="dt" %>
 ....
 <do type="options" name="h" label="Help">
 <go href="<dt:getHelp/>"/>
 </do>
```

Native Portal Desktop Templates

The template desktop automatically populates a help option in the channel if the template desktop finds an appropriate `helpURL` property for the channel in the display profile.

See the template `provHelpLink.template` and `frontHelpLink.template`.

Updating the Channel Display Profile

You must update the display profile for the channel to include a `helpURL` property that is specific to the client devices you are providing help for. To specify a display profile property specific to a client device (or a group of devices), use conditional properties.

To update a channel's display profile, you must add a custom channel property and define its characteristics.

To Set Up a Custom Channel Property

1. Log in to the Identity Server administration console as the administrator.

   By default, Identity Management is selected in the Header frame (the top horizontal frame) and Organizations is selected in the View menu of the Navigation frame (the left vertical frame).

2. From the View menu, click Services.

3. Click the arrow for Portal Desktop under the Portal Server Configuration heading.

   The Portal Desktop page appears in the Data frame (the right vertical frame).

4. Click the Manage Channels and Containers link for the Display Profile.

   The Channels page appears in the Data frame.

5. Scroll down to the Channels Section and click the Edit Properties link for the channel that you want to add Help to (for example, `UserInfo`).

   The Edit Channel Properties page for the selected channel is displayed.

6. Click New to add a custom channel property.

7. In the New Property page:

   a. From the Type menu, click Conditional Property.

   b. In the Condition field, enter **`client`**.

   c. In the Value field, enter the client type for the device you are providing Help for (for example, **`WML`**).

   d. Click Save (or click Save and Add Another if you want to add another property).

      The Edit Channel Properties page appears again with your new property added to the list.

To Define a Custom Channel Property

NOTE If you are not already logged in, log in to the Identity Server administration console as the administrator. By default, Identity Management is selected in the Header frame (the top horizontal frame) and Organizations is selected in the View menu of the Navigation frame (the left vertical frame).

1. From the View menu, click Services.
2. Click the arrow for Portal Desktop under the Portal Server Configuration heading.

   The Portal Desktop page appears in the Data Frame (the right vertical frame).
3. Click the Manage Channels and Containers link for the Display Profile.
4. Click the Edit Properties link for the channel that you want to add Help to.

1. Click the appropriate link. For example, for the WML client type, click the "client=WML" link.

   The appropriate client type window appears, such as the "client=WML" window.
2. Click the New link to add a custom channel property.
3. From the Type menu, select String.
4. In the Name field, enter **`helpURL`**.
5. In the Value field, enter the location of your Help file relative to your Portal Server web application documentation root (for example, `mobile/wml/help.wml`).
6. Click Save (or click Save and Add Another if you want to add another property).

   The location of your Help file is now displayed under the Customized heading in the `helpURL` field.

Creating a Help File

Create the help file using markup language appropriate to the client, or use AML for the rendered Portal Desktop. If you create an AML help file, the rendering engine processes and renders the file appropriately for the mobile client.

NOTE To be processed by the rendering engine, the AML help file must be a JavaServer Pages specification (for example, `help.jsp`), and `contentType` must be set to `text/aml`.

To do this, insert the following line at the top of the JSP help file:

```
<%@ page contentType="text/aml" %>
<AmlDocument>
....
```

Installing Help Files in the File System

Once you have created the help file, copy the file to the web application documentation root for Portal Server software:

/portal-server-installation-root`/SUNWps/web-src/docs`

NOTE To be processed by the rendering engine, AML help files must be stored under Portal Server software's web application documentation root (for example, `/opt/SUNWps/web-src/docs`).

Deploying Help Files

Deploy the help file using Portal Server software's deploy command:

/portal-server-installation-root`/SUNWps/bin/deploy redeploy`

Glossary

See the Java Enterprise System Glossary (http://docs.sun.com/doc/816-6873) for a complete list of terms that are used in this documentation set.

Index

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X