BEA WebLogic Mobility Server

User Guide

Version 3.6 SP1
Oct 2007

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

Introduction

About this Manual

This manual introduces you to the various features of BEA WebLogic Mobility Server™ and explains how to create a web site that targets all devices.

The manual is divided into eight parts:

Part I – Documentation Roadmap

This section provides an overview of where to look for information as you develop your mobilized web applications.

Part II – The Mobility Extension/Plugin Features

This section gives an overview of the features provided by the Mobility Extension/Plugin and introduces the concept of Mobility tags.

Part III – Fundamentals of Mobile Content

This section introduces the WebLogic Mobility Server mobility tags and describes key fundamentals behind working with mobile content, including organizing content and creating conditional content. It provides an overview of the process used in creating web applications that target a variety of devices.

Part IV – Presentation of Mobile Content

This section describes key aspects of content presentation, including layouts and structures, server-side style sheets, navigation, tables and media.

Part V – The Delivery Context API

This section introduces the delivery context API and explains how to use its methods to obtain device information from the WebLogic Mobility Server database.

Part VI – Diagnostics

This section describes the WebLogic Mobility Server diagnostic tools that enable developers and administrators to monitor the HTTP request/response cycle within WebLogic Mobility Server and to retrieve diagnostic information generated in the process.

Part VII – Glossary

This section provides an explanation of terms and acronyms used in this document.

Part VIII – Appendixes

This section provides a reference guide to the WebLogic Mobility Server mobility tags, a delivery context API reference guide, a list of deprecated items, and answers to frequently asked questions.

Prerequisites

This manual assumes that you are familiar with web page design using HTML and XHTML. It also assumes that you are familiar with basic JSP (or related scripted web page technologies) and the use of basic Java within such pages to generate dynamic content.

The manual does not teach specific web or site design skills, but will show you how to use WebLogic Mobility Server to bring mobility into your existing web design workflow.

Terminology Used in this Manual

There are a number of terms used throughout this manual that have a specific meaning within WebLogic Mobility Server:

- The term *content* describes the elements that define your service: XHTML mark-up (such as tables and formatting), JavaScript, JSP code, text, images and so forth.
- *Static content* refers to web pages that are sent to the requesting browser without changes made by scripting or code. It is usually information that does not need to change on a regular basis.
- *Dynamic content* describes web pages that are generated when accessed by an end-user, possibly pulling in external content and/or pages customized for the requesting user.
- The term *request page* refers to the file containing the content that will be transformed.
- The term *web application* refers to a web-enabled application that consists of static and dynamic resources, including multiple servlets and JSPs.
- The term *device* should be taken to refer either to a specific device, such as an iPAQ, or a device category, such as menu-driven devices or PDAs.
- The term **<WLMS_install_directory>** refers to either **<BEA_install_directory>\weblogic81\mobility**, **<BEA_install_directory>\weblogic92\mobility** or **<BEA_install_directory>\wlserver_10.0\mobility** depending on your installation

Content Examples

This manual describes the two mobility mark-up tag sets. The code examples in this manual use either the mmXHTML tags or their equivalent JSP taglib. In most cases the examples can be done using either set.

Further Reading

The following documentation set is provided with WebLogic Mobility Server:

- *BEA WebLogic Mobility Server and Device Repository Documentation Roadmap*
- *BEA WebLogic Mobility Server Installation Guide*
- *Device Repository Guide*
- *BEA WebLogic Mobility Server Administration Guide*
- *BEA WebLogic Mobility Server Getting Started Guide*
- *BEA WebLogic Mobility Server User Guide*
- *BEA Sample Workshop Mobility Project Guide*
- *BEA Sample Mobility Portal Guide*
- *BEA Mobilize Your Portal Guide*

Part I Documentation Roadmap

This section provides an overview of where to look for related information as you develop your mobilized web application.

Where Do I Start?

After installation, we recommend you begin with the *Getting Started Tutorials*. The tutorial set is a series of short lessons that introduce the WebLogic Mobility Server mobility tags and the delivery context API. The lessons are made up of short, working examples that take you through the process of marking up content in order to create web pages that can be viewed on many different client devices. You also learn how to use the delivery context API, which lets developers obtain device attribute information from the Device Repository. This is helpful when creating dynamic content tailored to the specific requesting device.

What Manuals Are Provided?

All the WebLogic Mobility Server manuals are supplied in Adobe Acrobat format. The following table describes the manuals available and their purpose:

WebLogic Mobility Server Documentation Set

Document	Description
BEA WebLogic Mobility Server and Device Repository Documentation Roadmap	Briefly explains how to use the set of manuals provided with WebLogic Mobility Server and the Device Repository it uses.
BEA WebLogic Mobility Server Installation Guide	Provides instructions for installing and configuring WebLogic Mobility Server.
Device Repository Guide	This guide explains how to install the Device Repository used by WebLogic Mobility Server, describes how to update the *mis.properties* file to reflect the Device Repository connection details and outlines how to set up and manage the device profiles stored in the Repository.
BEA WebLogic Mobility Server Administration Guide	Provides information on how to configure WebLogic Mobility Server after installation.
BEA WebLogic Mobility Server Getting Started Guide	A tutorial manual designed to get you started with the WebLogic Mobility Server mobility tags by stepping through a series of short, self-contained projects.
BEA WebLogic Mobility Server User Guide (this guide)	Provides supporting and advanced information on WebLogic Mobility Server, including: • The general principles behind organization and transformation of content. • Information on using the WebLogic Mobility Server diagnostics

	to monitor internal behavior during transactions in development and production environments. • An appendix section on the WebLogic Mobility Server mobility tags and the delivery context API.
BEA Sample Workshop Mobility Project Guide	Provides a step-by-step exercise in mobilizing a sample workshop application on WebLogic 8.1 platforms
*Sample Mobility Portal Guide**	Explains the features of the mobilized BEA Portal Framework using a sample Mobility Portal
*Mobilize Your Portal Guide***	Explains how to mobilize a portal and then apply your own "Look & Feel" to it as required.

Is There a Demonstration Workshop Mobility Project?

Yes, if you have installed BEA WebLogic Mobility Server available from BEA Systems on BEA WebLogic 8.1. The "restaurantWeb" sample 8.1 Workshop project demonstrates many of the WebLogic Mobility Server features such as:

- Organizing and tailoring content for different devices
- Previewing content on different devices
- Ensuring the media format being delivered is appropriate to the device making the request

The "restaurantWeb" sample Workshop project has been installed in the **<WLMS_install_directory>\samples\BEAWorkshop** directory. Please see the *BEA WebLogic Mobility Server Installation Guide* for instructions on importing the "restaurantWeb" sample Workshop project into a Workshop application. The *BEA Sample Workshop Mobility Project Guide* provides a step by step explanation of how to use the Mobility Extension for BEA WebLogic Workshop 8.1 to mobilize "restaurantWeb." In the same directory as "restaurantWeb" is an already mobilized version of the sample project called "restaurantWeb_after."

What Authoring Tools Do I Need?

WebLogic Mobility Server includes the Mobility Extension for BEA WebLogic Workshop. With the Mobility Extension, the BEA WebLogic Workshop Integrated Development Environment can be used for creating multi-channel applications and extending existing applications to support wireless devices. Please see the section "The Mobility Extension for BEA Workshop" for a description of the functionality provided by the Mobility Extension. The *BEA Sample Workshop Mobility Project Guide* provides a hands-on tutorial that uses the Mobility Extension features.

What Format Do I Use for Marking Up Content?

The mobility tags are a set of XML compliant tags that form the essential building blocks for the development of mobilized web content.

mmXHTML (multi-mode XHTML) is a compact set of tags that begin with the characters mm- (for example `<mm-group>…</mm-group>`). The WebLogic Mobility Server JSP tag library is an equivalent set of tags for developers working with JSP pages. The taglib replicates the functionality of the mmXHTML tag set. These tags begin with the characters mm: (for example `<mm:group>…</mm:group>`). The Mobility Extension for BEA WebLogic Workshop integrates

the JSP tab library into WebLogic Workshop, providing a Mobility Palette and inserts wizards for drag and drop inclusion of the mobility tags into content.

How Do I Change the Appearance of Web Content for Mobile Devices?

WebLogic Mobility Server uses the concepts of layouts and structures to control the organization and transformation of content requested by mobile devices. The `<mm-structure>` and `<mm-layout>` tags are responsible for restructuring a single source of web content for presentation on a variety of handheld devices that require special consideration because of screen size and memory limitations. The `<mm-layout>` tag allows the user to specify a file that contains alternate, simplified templates for PDA and menu-driven devices. The `<mm-structure>` tag is used to redisplay the content for smaller devices. It gives the author the power to control the navigation flow between pages, which is often needed to present content clearly on these smaller devices.

Where Can I Find More Information on Delivering Images and Other Media to Mobile Devices?

WebLogic Mobility Server provides the `<mm-media-group>` and `<mm-img>` tags to ensure the appropriate image, in terms of size, quality and suitability, is delivered to a device. There is also an `<mm-logo>` tag that is used to flash an image on WML devices for a short time before presenting the rest of the content.

How Do I Tailor Content for Different Devices?

In addition to allowing the creation of specific layouts for targeted device groups, WebLogic Mobility Server provides the `<mm-include>` and `<mm-exclude>` tags to conditionally include and exclude parts of the content depending on either the type of device making the request or specific device profile attributes.

How Can I Create Dynamic Content That Accommodates Device Characteristics?

You can use JSP to output WebLogic Mobility Server mark-up that accommodates different devices. For example, you might want to change the number of columns in a table depending on the width of the device's screen. To do this, you would need specific information from the Device Repository. By using the delivery context API, you could retrieve the required device attributes, which could then be placed in the JSP to dynamically change what is delivered to the device.

How Do I Brand or Use Logos on my Device?

For targeting WML devices, you can take advantage of a feature that displays a logo for a short period of time before rendering the rest of the content. Use the `<mm-logo>` tag for this purpose.

How Do I Troubleshoot the Output From WebLogic Mobility Server?

WebLogic Mobility Server provides diagnostic tools to monitor a range of system messages, which allows you to troubleshoot different phases of the transformation process. The diagnostic messages show the content before and after it is transformed, as well as the processing that occurs in-between. You can select a variety of diagnostic "topics" in order to target specific areas to monitor.

How Can I Modify the Basic Start-Up Parameters for WebLogic Mobility Server?

Configuration settings for WebLogic Mobility Server are stored in the **WEB-INF/classes/***mis.properties* file. This file contains settings that enable WebLogic Mobility Server to communicate with the Device Repository as well as updating log files and other transaction-related activity. The *BEA WebLogic Mobility Server Administration Guide* describes the settings in this file.

How Do I Take Advantage of Access Keys on WML Devices?

Access keys provide a shortcut for navigating content delivered to menu-driven devices that support this feature.

How Do I Find Out More About Working with Tables?

Mobile devices differ in their ability to handle tables. WebLogic Mobility Server supplies the mobility tag <mm-table-model> for managing the transformation of tables. See the section “Work with Tables” for additional information.

Part II The Mobility Extension/Plugin Features

Important note: This chapter is split into two sections—please see either section “The Mobility Extension for BEA WebLogic Workshop 8.1” or section “The Mobility Plugin for BEA Workshop for WebLogic Platform 9.2/10” according to your authoring environment.

The Mobility Plugin for BEA Workshop for WebLogic Platform 9.2/10

Important note: This section is only relevant if you are using WebLogic Mobility Server in a BEA Workshop for WebLogic Platform 9.2/10 authoring environment.

After installing WebLogic Mobility Server 3.6 with the Mobility Plugin, the Integrated Development Environment (IDE) can be used for creating multi-channel applications and extending existing applications to wireless.

Summary of the Mobility Plugin Features

This section provides an overview of the Mobility Plugin features, including:

- The Enable Multi-Channel feature for new or existing projects
- The Mobility right-click menu, for easy access to the Mobility tags
- The Mobility toolbar, including:
  - Design Preview icons for the different device classes
  - Emulator Launch icons for the different device classes
  - Launch Mobility Tools icon
- Window Preferences panel additions, including:
  - Mobility Pane, for setup and configuration
  - Emulators Pane, for configuring Device Emulators
- The Mobility Plugin Help

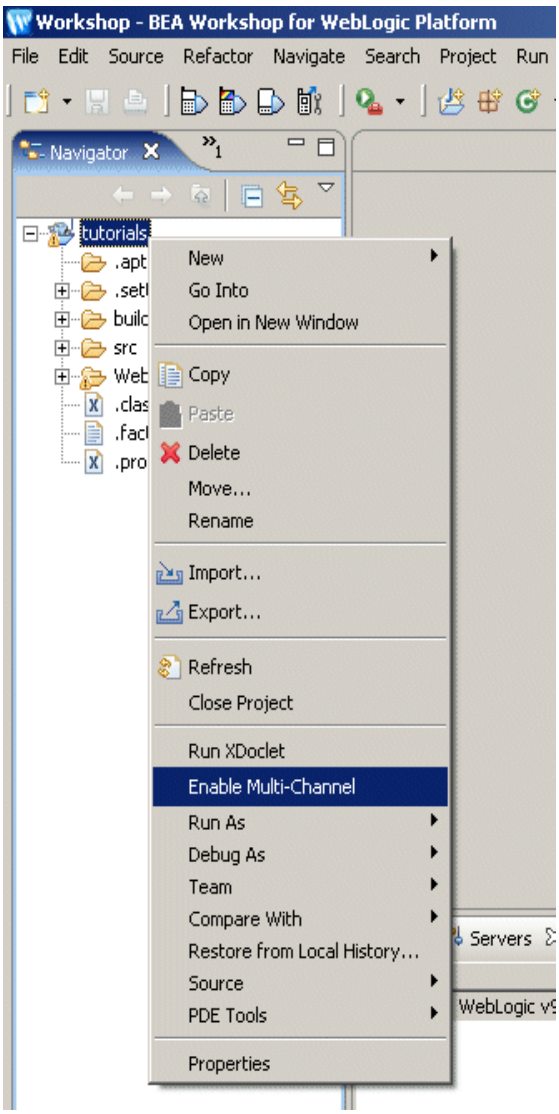
These features are described in more detail in the sections below.

Note: The large XML-format Device Repository file may cause problems when a project is opened in Workshop. Please use the compressed MADR-format repository to avoid these problems. The Enable Multi-Channel function automatically adds the ".madr" version of the file to your project.

Enable Multi-Channel

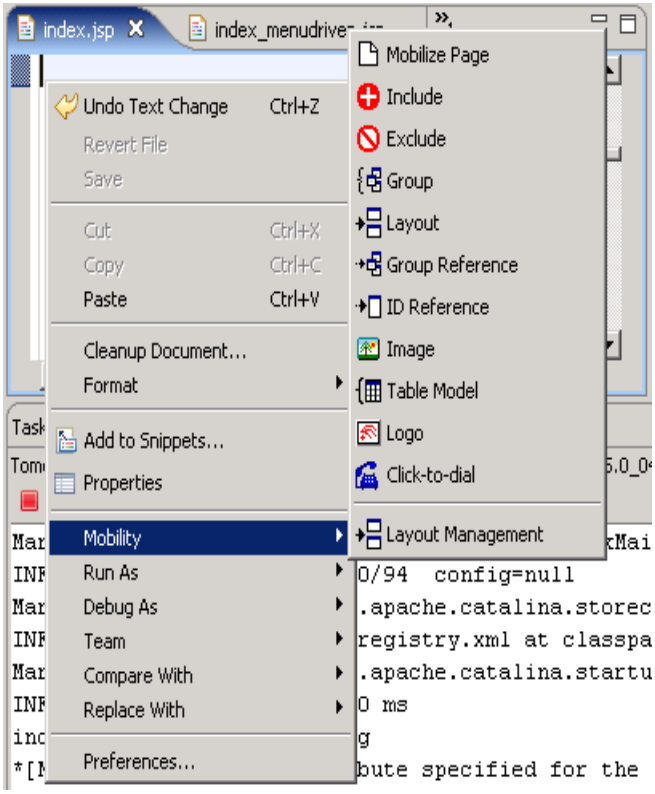
Any application project can be configured for multi-channel delivery by selecting the “Enable multi-channel” option for the highlighted project. Selecting this option configures the application project to deploy and use WebLogic Mobility Server.

Enable Multi Channel



Mobility Right-Click Menu

Right-click in the Editor pane to display the Mobility menu, which allows you to apply mobility meta-data to any HTML or JSP content to enable delivery to mobile devices. This includes support for wizard-based generation of layouts for different device categories, click-to-dial for embedding telephony commands in applications, multi-device image handling, and device-category styling.



Mobility Toolbar

This is a toolbar extension that enables launching of device emulators for smart phones and PDAs in addition to the test browser. The Mobility toolbar also provides a diagnostics application that enables the simulation of mobile requests for detailed testing of applications.

The toolbar additionally provides the capability to manage target devices from within the authoring environment, enabling addition/removal of devices in the Device Repository and device profile configuration.

Viewing Content with Emulators

The Mobility Plugin installation adds a toolbar that makes it easy to view and troubleshoot mobilized content. Once your emulators have been configured as described in the installation instructions, you will have an extra toolbar in the authoring environment, the Mobility toolbar.

The first three icons let you launch the particular device emulator to see the results of the mobility markup, as it would appear in the actual device.

They are, in order:

Launch Configured WAP 1.x Emulator

Launch Configured WAP 2.x Emulator

Launch Configured PDA Emulator

Launch Other Configured Emulators / Browsers

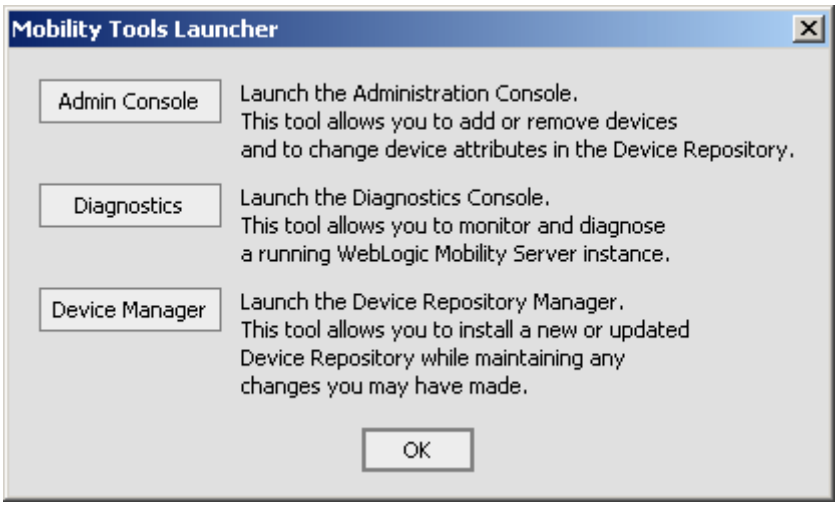
The sample project, when viewed in a menu-driven (for example smart phones) or a PDA device, demonstrates the types of issues that can arise when sending PC web content to handheld devices. As the code is mobilized, you will be able to see how WebLogic Mobility Server can manipulate the content to make it look good on all types of devices.

The final icon on the toolbar gives access to additional WebLogic Mobility Server tools:

- Administration Console for managing devices and their attributes
- Diagnostics Console for diagnostic output and troubleshooting
- Device Repository Manager for managing the Device Repository

Clicking on this icon will open the Mobility Tools Launcher dialog box for these tools (see the following graphic).

Tools Launcher



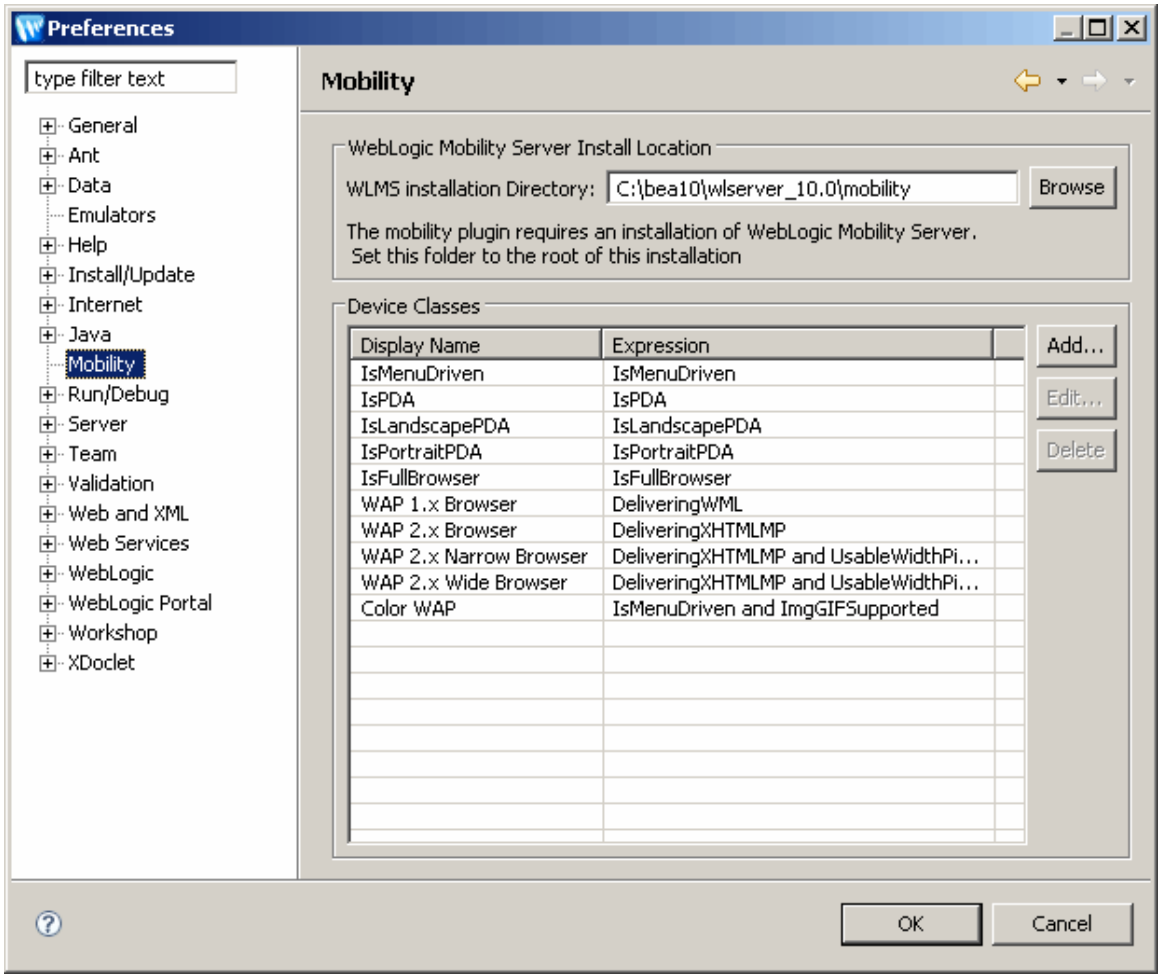
Preferences Additions

Two additional panes have been added to the Window Preferences dialog (choose **Window → Preferences**):

- Mobility pane
- Emulators pane

Mobility Pane

The “Mobility” pane allows the user to set the WebLogic Mobility Server Install Location property.

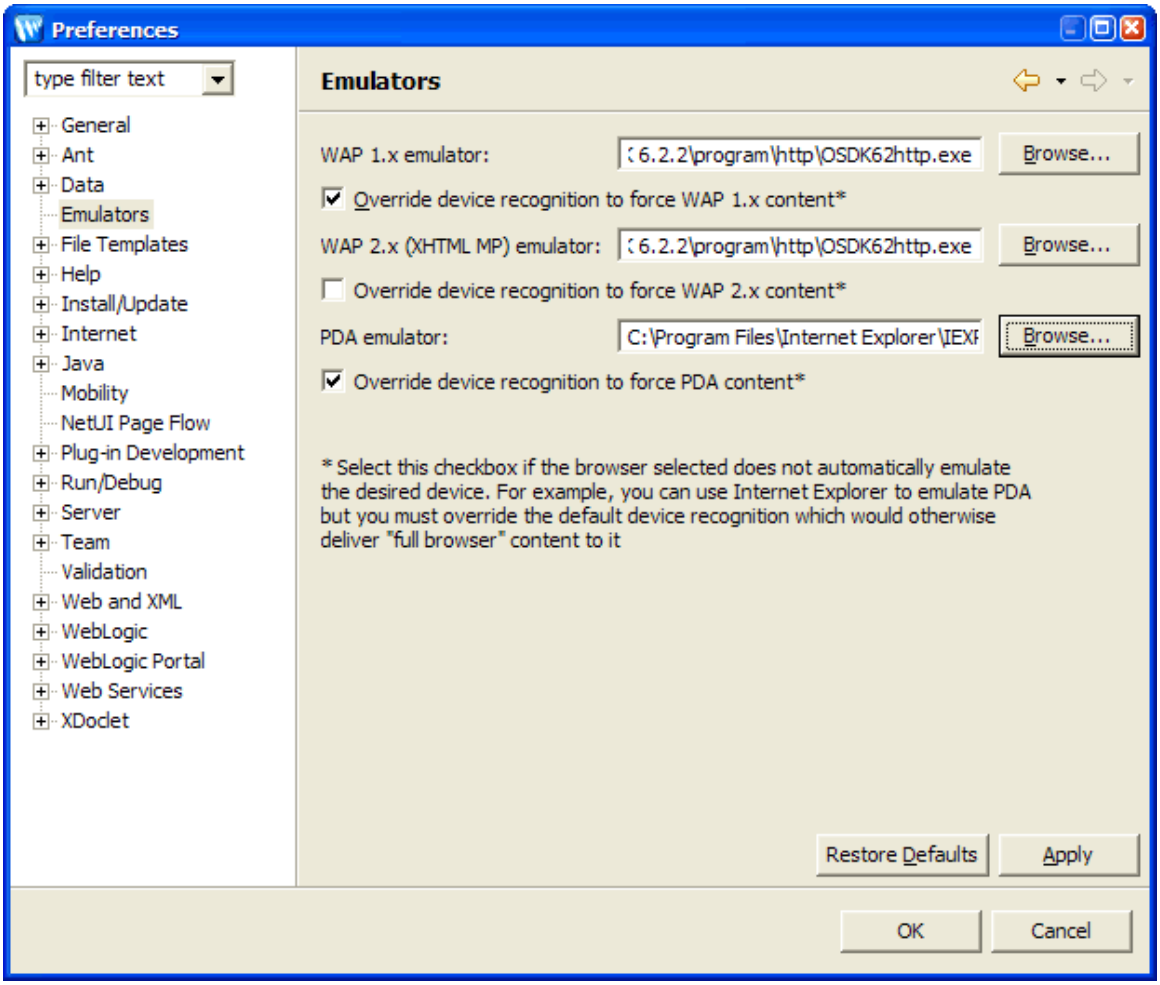


The “Device Classes” section provides a list of configured device classes when using the Mobility tags. Developers can specify frequently used “where” expressions for inclusion in the list. See chapter “Part III”, section “Create Conditional Content” for a detailed explanation of the Mobility tags and “where” expressions.

To add a new item to the drop down list, click **Add** and insert the frequently used expression in the **Expression** field and an easily identifiable display name to be associated with it in the **Display Name** field. Subsequent usage of the Mobility tags will allow the user to select this “Display Name” resulting in the automatic insertion of the defined “Expression”.

Emulators Pane

The “Emulators” pane enables configuration of different emulators for testing the application appearance and flow on a range of devices.



For more information on configuring these emulators, see the section “Configure the Device Emulators for use within…” in the *BEA WebLogic Mobility Server Installation Guide.*

The Mobility Extension for BEA WebLogic Workshop 8.1

Important note: This section is only relevant if you opted to install WebLogic Mobility Server on BEA WebLogic 8.1.

After installing WebLogic Mobility Server including the Mobility Extension for BEA WebLogic Workshop 8.1 (see the *BEA WebLogic Mobility Server Installation Guide*), the BEA WebLogic Workshop Integrated Development Environment can be used for creating multi-channel applications and extending existing applications to wireless.

Summary of the Mobility Extension Features

This section provides an overview of the Mobility Extension features, including:

- Enable Multi-Channel for new or existing projects
- Mobility Palette, for easy Access to Mobility Tags
- Mobility Toolbar, including:
  - Design Preview icons for the different device classes
  - Emulator Launch icons for the different device classes
  - Launch Mobility Tools icon
- IDE Properties additions, including:
  - Mobility Pane, for setup and configuration
  - Emulators Pane, for configuring Device Emulators
- Mobility Menu under Tools, including:
  - Launch Emulator options
  - Launch Mobility tools

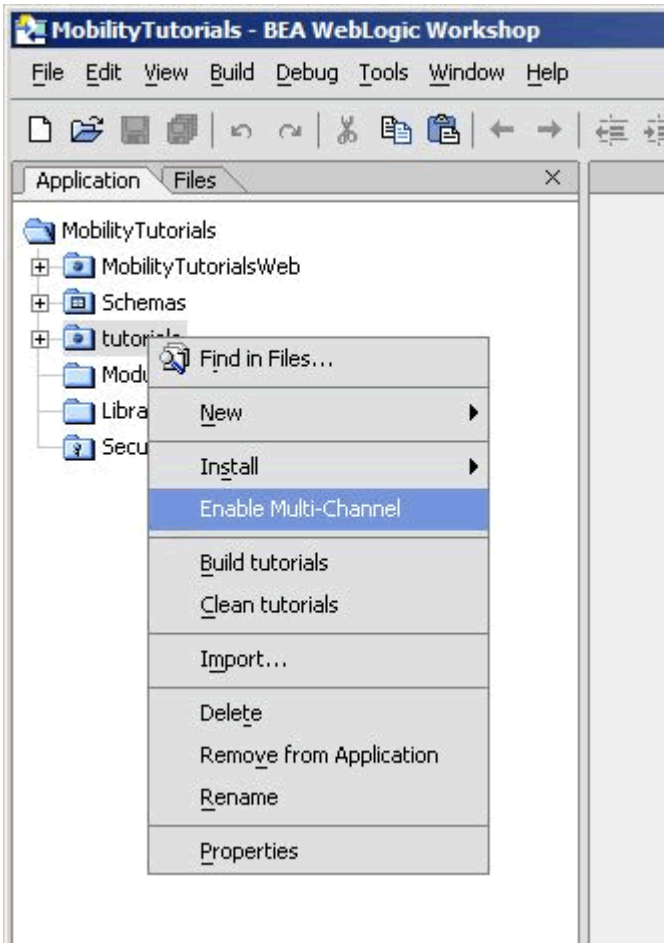
These features are described in more detail in the “Enable Multi-Channel” section.

Note: The large XML-format Device Repository file may cause problems when a project is opened in WebLogic Workshop. Please use the compressed MADR-format repository to avoid these problems. The Enable Multi-Channel function automatically adds the ".madr" version of the file to your project.

Enable Multi-Channel

Any application project can be configured for multi-channel delivery by selecting the “Enable multi-channel” option for the highlighted project. Selecting this option configures the application project to deploy and use WebLogic Mobility Server.

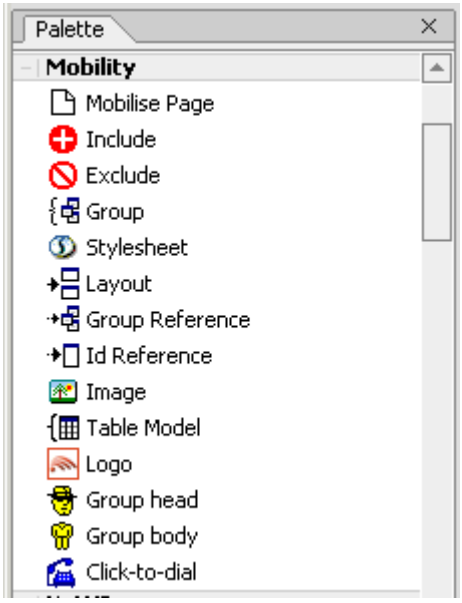
Enable multi-channel



Mobility Palette

Using a “drag & drop” technique, mobility meta-data can be applied to any NetUI-defined interface to enable delivery to mobile devices. This includes support for wizard-based generation of layouts for different device categories, click-to-dial for embedding telephony commands in applications, multi-device image handling, and device-category styling.

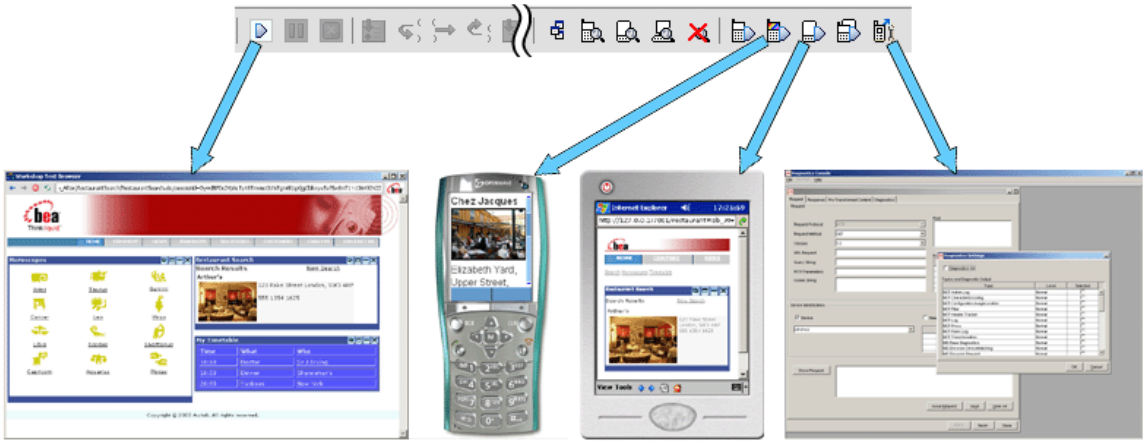
The Mobility Palette



Mobility Toolbar

This is a toolbar extension that enables launching of device emulators for smart phones and PDAs in addition to the Workshop test browser. The Mobility Toolbar also provides a diagnostics application that enables the simulation of advanced mobile requirements for detailed testing of applications.

The Mobility Toolbar features



The Mobility Toolbar additionally provides the capability to manage target devices from within Workshop, enabling addition/removal of devices in the Device Repository and device profile configuration for optimization of NetUI for different device types.

View Content with Emulators

The Mobility Extension installation adds a toolbar to Workshop that makes it easy to view and troubleshoot mobilized content. Once your emulators have been configured as described in the installation instructions, you will have an extra toolbar in Workshop, the Mobility Toolbar.

The Mobility Toolbar

The first icon simplifies the management of groups of content that have been created on the page that is open in the Edit Pane.

Manage Groups

The next four icons allow you to see the mark-up for a particular device. Content groups that have been excluded for a particular device will not appear in the window when that device's button is pressed. The groups that are to be delivered to the device will be expanded so that you can see approximately how they would look.

These four icons are as follows:

Preview WAP Content

Preview PDA Content

Preview Full Browser (PC) Content

View All Contents and Mobility Mark-up

The next four icons let you launch the particular device emulator to see the results of the mobility markup, as it would appear in the actual device.

They are, in order:

Launch Configured WAP 1.x Emulator

Launch Configured WAP 2.x Emulator

Launch Configured PDA Emulator

Launch Other Configured Emulators/Browsers

The sample project, when viewed in a menu-driven (for example smart phones) or a PDA device, demonstrates the types of issues that can arise when sending PC web content to handheld devices. As the code is mobilized, you will be able to see how WebLogic Mobility Server can manipulate the content to make it look good on all types of devices.

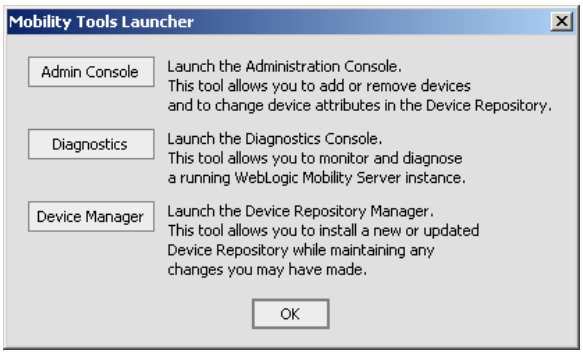
The final icon on the toolbar gives access to additional WebLogic Mobility Server tools:

- Admin Console for managing devices and their attributes
- Diagnostics Console for diagnostic output and troubleshooting
- Device Repository Manager for managing the Device Repository

WebLogic Mobility Server Tools

Clicking on this icon will open the Mobility Tools Launcher dialog box for these tools (see the following graphic). The tools can also be launched directly from the Mobility Menu.

Mobility Tools Launcher



IDE Properties Additions

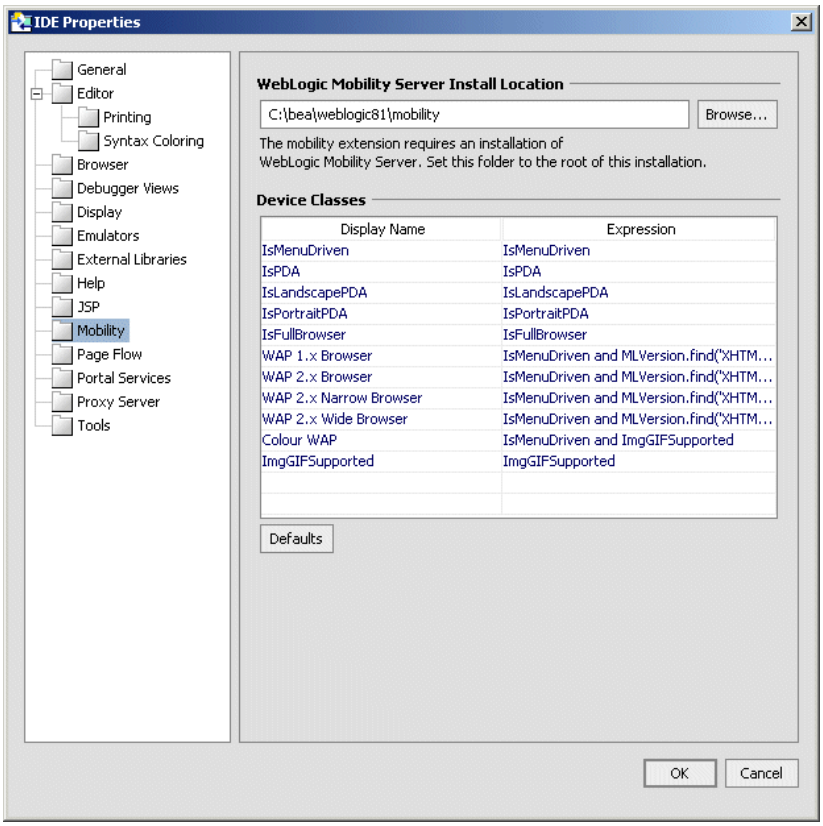
Two additional panes have been added to the IDE Properties dialog:

- Mobility pane
- Emulators pane

Mobility Pane

The "Mobility" pane allows the user to set the WebLogic Mobility Server Install Location property.

Additions to IDE Properties Dialog



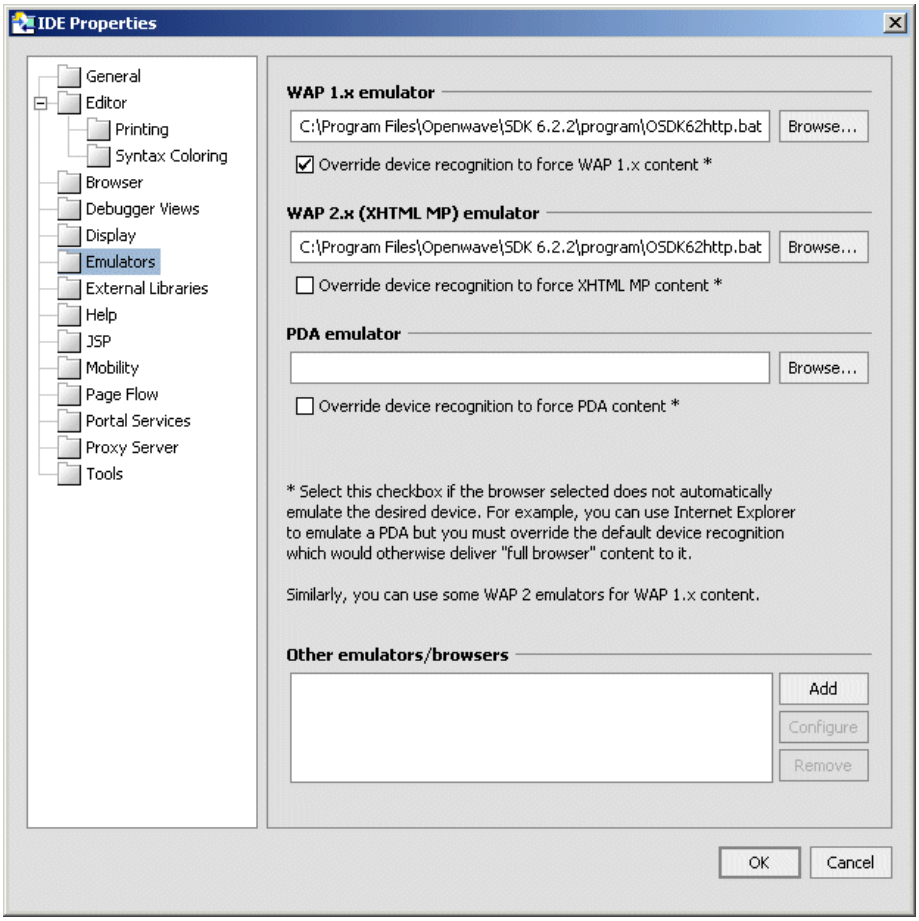
The "Device Classes" section provides a drop down list of configured device classes when using the Mobility Tags. Developers can specify frequently used "where" expressions for inclusion in the drop down list.

To add a new item to the drop down list, insert the frequently used expression in the "Expression" column and an easily identifiable "Display Name" to be associated with it. Subsequent usage of the Mobility Tags will allow the user to select this "Display Name" resulting in the automatic insertion of the defined "Expression".

Emulators Pane

The “Emulators” pane enables configuration of different emulators for testing the application appearance and flow on a range of devices.

Emulators Pane



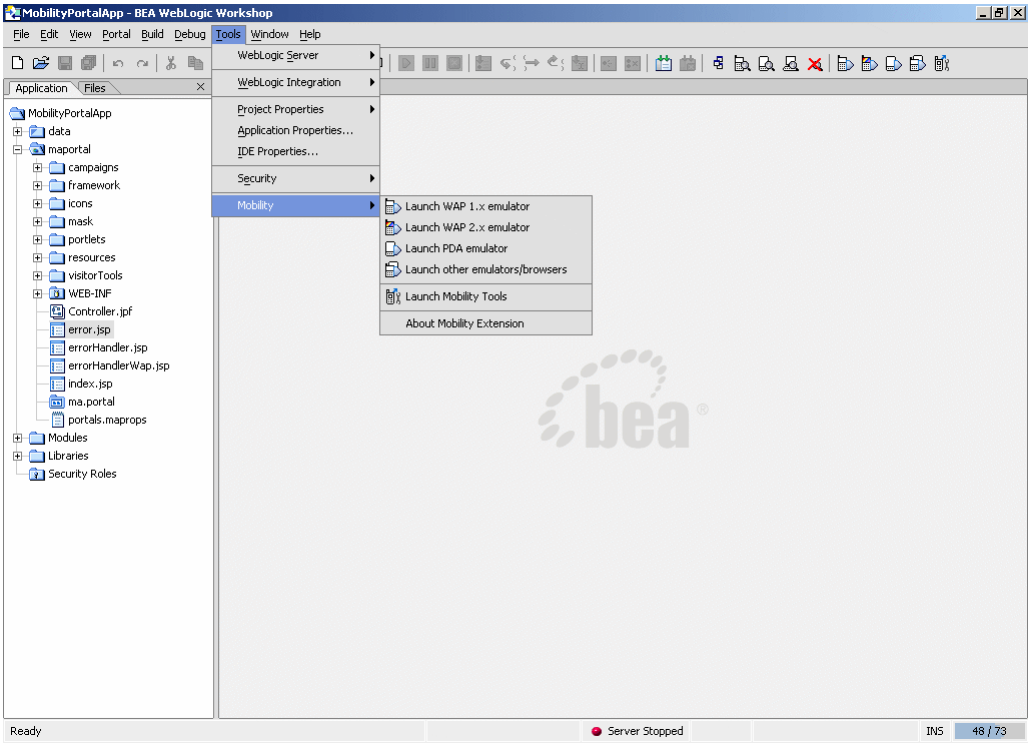
For more information on configuring these emulators, see the section “Configuring the Device Emulators” in the *BEA WebLogic Mobility Server Installation Guide.*

Mobility Menu

The Mobility Menu contains links to the following:

- Launch WAP 1.x emulator
- Launch WAP 2.x emulator
- Launch PDA emulator
- Launch other emulators/browsers
- Launch Mobility Tools

Mobility Menu



Part III Fundamentals of Mobile Content

Overview of the Process

This section provides an overview of the process used in creating web applications that target a variety of devices.

Three Steps to Creating and Delivering your Service

There are three basic steps in preparing content so that it can be displayed on multiple devices:

- Creating your content in XHTML format.
- Organizing the page content into logical divisions.
- Deciding how you want your content to be presented on different devices.

Create your content in XHTML format

All documents that are processed by WebLogic Mobility Server must follow XHTML 1.0 standards. XHTML has stricter mark-up standards than HTML. All tags, for example, must terminate correctly, either with a closing tag, or, for empty tags, by putting a closing slash before the final angle bracket.

All documents that are processed by WebLogic Mobility Server must follow XHTML 1.0 standards. XHTML has stricter mark-up standards than HTML. All tags, for example, must terminate correctly, either with a closing tag, or, for empty tags, by putting a closing slash before the final angle bracket.

If you are converting legacy content containing HTML mark-up, there are a number of tools available to automate the process of converting HTML to XHTML. The Developer Toolkit on CD2 includes *HTML Tidy*, a popular tool for converting HTML files to XHMTL. Dreamweaver MX also provides built-in XHTML clean-up and conversion utilities.

You can use any authoring environment to develop your XHTML content. This content can be either *static*, where the page does not change, or *dynamic*, where you use JSP to dynamically generate content.

Organize the page content into logical divisions

Since the amount of content you can display depends on screen size, you will want to divide your content into logical units called *groups*. This will give you more flexibility in deciding what gets displayed on specific devices. You can take advantage of the natural divisions in your document (such as tables and forms) or you can explicitly create your own groups. For example, you might arrange a news page into national, regional and local news groups.

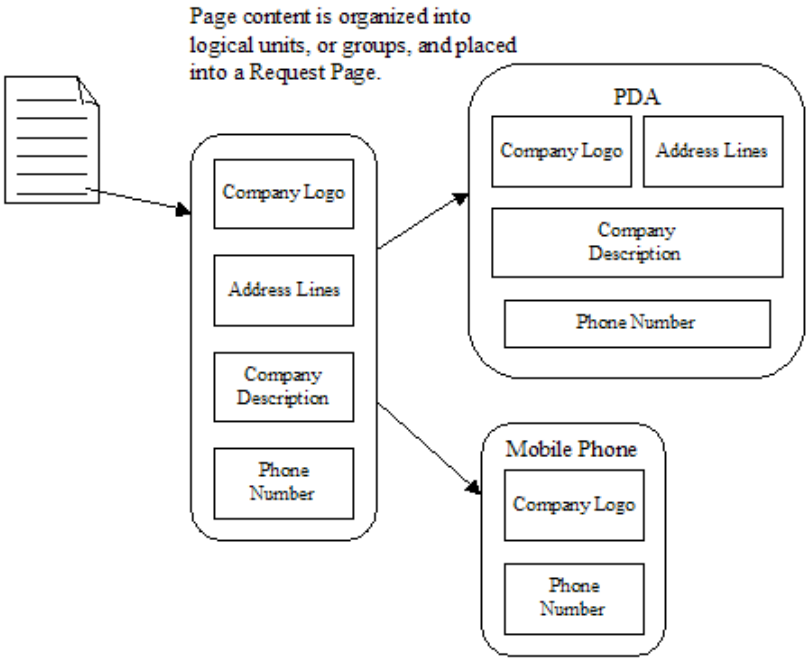
Decide how you want your content to be presented on different devices

You can control which groups get sent to the different device types. You can also control the way the content is organized for each type of device. The different screen sizes and memory capabilities of the handheld devices often require that the content be arranged differently to best suit capabilities of the requesting device. This is achieved through the use of *layout files* that instruct WebLogic Mobility Server how to present the content for these smaller devices. For example, one layout for a PDA device might extract and display a company logo, an address group, a phone group and a text group describing the company. Another layout, for a smaller screen, might just extract and display the company logo group and a phone number

You can control which groups get sent to the different device types. You can also control the way the content is organized for each type of device. The different screen sizes and memory capabilities of the handheld devices often require that the content be arranged differently to best suit capabilities of the requesting device. This is achieved through the use of *layout files* that instruct WebLogic Mobility Server how to present the content for these smaller devices. For example, one layout for a PDA device might extract and display a company logo, an address group, a phone group and a text group describing the company. Another layout, for a smaller screen, might just extract and display the company logo group and a phone number.

The following illustration shows how three different devices might organize the same content.

Content reorganized on a variety of devices



Introducing the Mobility Tags

This section introduces the mobility tags; the WebLogic Mobility Server authoring extensions to XHTML. It gives a brief background of XHTML and shows how the mobility tags extend and conform to this schema.

The History of XHTML

HTML was originally designed for publishing hypertext on the World Wide Web. HTML is a non-proprietary format based on SGML. Variations in HTML language definitions have emerged in the form of browser-specific implementations. These variations, combined with the increasing availability and diverse capabilities of non-desktop browsers, such as PDAs and mobile phones, have made it difficult for designers to settle on a system for targeting a range of devices. Many developers found themselves having to take a "lowest common denominator" approach when trying to combine content, physical layout and interactive behavior of a web application designed for a broad spectrum of client devices.

XHTML (Extensible Hypertext Mark-up Language), created by the World Wide Web Consortium W3C, addresses this issue. XHTML takes advantage of the more powerful meta-language XML and thus requires that documents be “well-formed”. Variations are not allowed. The strict mark-up standards are what allow extensibility of the core tag set to extend functionality.

XHTML 1.0 has replaced HTML as the official Web mark-up standard according to the W3C. XHTML is also the content authoring language specified by WAP 2.0 and XHTML Mobile Profile, used by many handheld devices, is a subset of XHTML.

XHTML looks a lot like HTML but has more rigid mark-up standards. XHTML documents must strictly conform to these standards, that is, be “well-formed”. For more on these standards and how to convert existing HTML to XHTML, see working with XHTML in this manual.

Extend XHTML with the Mobility Tags

WebLogic Mobility Server has added a module to XHTML, called *multi-mode XHTML***,** or *mmXHTML***.** This module is a set of XML elements that simplifies the task of structuring and presenting web applications on a range of devices including PCs, PDAs and web-enabled phones.

By adding these tags to “well-formed” web content, authors can create documents that can be displayed on a variety of devices. The WebLogic Mobility Server transformation engine interprets the tags and translates the XHTML content into the language understood by the requesting device. The mmXHTML tags also give content authors the power take a single source of content and re-organize and style the layout in a ways that best suit various desktop and mobile devices.

A JSP tag library version of the mmXHTML tags is also available for use with JSP files. These tags are almost identical in syntax and function. They are explained further in The WebLogic Mobility Server JSP Tag Library in this manual.

The Mobility Tags

The following table summarizes the mobility tags:

Mobility Tags

Element	Usage
mm-body	Used as a container for the content within an mm-group element.
<![CDATA[…]>	Used to hide content from the WebLogic Mobility Server parser.
mm-exclude	Used as a container to exclude content intended for certain devices but not others.
mm-group	Used to explicitly organize content into logical groups so that they can be referenced from different layouts that cater to different devices.
mm-group-ref	Used to reference an existing mm-group element.
mm-head	Used to specify a heading within an mm-group element to give context to content delivered to smaller devices that may require it to be broken into several pages.
mm-id-ref	Used to reference a group of content. Usually used inside a layout file.
mm-img	Used to deliver the correct image based on the capabilities of the target device. Can be placed inside a media-group.
mm-include	Used as a container to include content intended for certain devices but not others.
mm-layout	Used to specify the file that contains alternate layouts for handheld devices.
mm-li	Used with <mm-nl> to create easily styled navigation menus for handheld devices.
mm-logo	Used to flash an image (on WML devices that support this feature) for a few seconds before delivering the main content.
mm-media-group	Used to contain mm-img tags that refer to images of different sizes and formats. WebLogic Mobility Server will select the best image for the requesting device.
mm-nl	Used with <mm-li> to create easily styled navigation menus for handheld devices.
mm-phone-number	Used to include a dialable phone number link in content for WTAI enabled devices.

mm-structure	Used to contain references to groups that are to be delivered to menu-driven devices. This tag can also be used to create navigational menus for content that is delivered to PDAs.
mm-table-model	Used to instruct WebLogic Mobility Server to transform tables for small devices.

The mmXHTML Document Type Definition (DTD)

DTD Function

The purpose of the mmXHTML Document Type Definition (DTD) is to allow authors to ensure that documents are valid for use with WebLogic Mobility Server.

The DTD includes the language elements and attributes of WebLogic Mobility Server. It defines how tags and attributes are used to describe content in an mmXHTML document, where each tag is allowed, and which tags can appear within other tags.

For example, in the mmXHTML DTD `<mm-structure>` tags can contain `<mm-group-ref>` tags, but `<mm-group-ref>` tags cannot contain `<mm-structure>` tags.

DTD Usage

A reference to the DTD must be included in any mmXHTML document processed by WebLogic Mobility Server. When an author is creating an mmXHTML document, the document will require an XML declaration:

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

and a reference to the mmXHTML DTD within which the schema of the mobility tags are defined:

```
<!DOCTYPE html PUBLIC "-//MOBILEAWARE//DTD MMXHTML 1.2//EN"
"http://www.mobileaware.com/DTD/mmxhtml_1.2.dtd">
```

Note*:* WebLogic Mobility Server DTD is versioned. All future DTDs will be versioned also.

Sample Document with DTD

Here is an example file using the WebLogic Mobility Server DTD.

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//MOBILEAWARE//DTD MMXHTML 1.2//EN"
"http://www.mobileaware.com/DTD/mmxhtml_1.2.dtd">
<html>
    <head>
        <title>Demonstrating use of the WebLogic Mobility Server DTD</title>
    </head>
    <body>
        <p>The DTD reference is placed below the XML file declaration</p>
    </body>
</html>
```

Note: JSP files using the WebLogic Mobility Server JSP tag library require the WebLogic Mobility Server JSP document heading instead of the DTD declaration.

The WebLogic Mobility Server JSP Tag Library

The multi-mode JSP tag library (taglib) was introduced for the mobilization of JSP documents. The functionality of the mmXHTML tags is replicated within a JSP tag library structure. The taglib, however, provides a particular benefit over mmXHTML by adding some further optimizations in the performance of WebLogic Mobility Server when delivering JSP content to the PC channel.

This section covers the following aspects of the WebLogic Mobility Server JSP taglib:

- Enhanced Performance
- Syntax
- Mixing mm-tags and mm:tags
- Exception Handling and Diagnostics

Enhanced Performance

Marking up content with the JSP mobility taglib results in speed benefits when content is delivered to PCs. The WebLogic Mobility Server JSP tags reduce the amount of run-time processing that WebLogic Mobility Server is required to do. The tags run within the servlet environment, unneeded tags are stripped off, and the streamlined file is sent on to WebLogic Mobility Server where a minimum of processing is required before it is sent out to the browser. The result is a faster overall content delivery time.

Syntax

The WebLogic Mobility Server JSP tags mimic their mmXHTML counterparts with the exception that the tag name is prefixed by "mm:" instead of "mm-".

In order for the tags to be processed, two things are needed at the outset:

- A taglib declaration at the beginning of the document.
- An `<mm:page>` element surrounding the rest of the file's contents.

```
<%@ taglib uri="mmJSPtaglib" prefix="mm"%>
<mm:page>
<html>
   <head> Using JSP taglib</head>
   <body>
      <p>JSP page content here.</p>
   </body>
</html>
</mm:page>
```

mmXHTML headers and tags and the JSP tag library equivalents

mmXHTML	JSP Tag Library
Document must start with XML declaration and DOCTYPE: <?xml version="1.0"?> <!DOCTYPE html PUBLIC "-//MOBILEAWARE//DTD MMXHTML 1.2//EN"	Document must start with taglib declaration and the page must be wrapped in an <mm:page> tag: <%@ taglib uri="mmJSPtaglib" prefix="mm"%> <mm:page> ...

"http://www.mobileaware.com/DTD/mmxhtml_1.2.dtd">	</mm:page>
mm-body	mm;body
mm-exclude	mm:exclude
mm-group	mm:group
mm-group-ref	mm:group-ref
mm-head	mm:head
mm-id-ref	mm:id-ref
mm-img	mm:img
mm-include	mm:include
mm-layout	mm:layout
mm-li	mm:li
mm-logo	mm:logo
mm-media-group	mm:media-group
mm-nl	mm:nl
mm-phone-number	mm:phone-number
mm-structure	mm:structure
mm-table-model	mm:table-model
For data that should be ignored by the MIS parser: <meta name="MIS-CDATA-Control" content="Unwrap"/> <![CDATA[...]]>	<mm:meta name="MIS-CDATA-Control" content="Unwrap"/> <mm:cdata>...</mm:cdata>
For the creation of an Option menu: <meta name=..." content="..." scheme="mmsection"/>	<mm:meta name="..." content="..." scheme="mmsection"/>

Examples

Here are two versions of a short file that will be transformed by WebLogic Mobility Server. They demonstrate the way in which the mobility tags are placed into a document. The meaning of the tags used in these examples will be explained in later sections. The first file uses standard mmXHTML and is saved as an *.htm* file. The second file uses the equivalent JSP tag libraries. It is stored as a *.jsp* file.

Example 1: hello.htm

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//MOBILEAWARE//DTD MMXHTML 1.2//EN"
 "http://www.mobileaware.com/DTD/mmxhtml_1.2.dtd">

<html>
   <head>
      <title>Hello World!</title>
   </head>
   <mm-structure id="structure_1" where="IsMenuDriven">
      <mm-group-ref idref="mobileGreeting" type="normal" depth="0"
       display="all"/>
   </mm-structure>
   <body>
      <p>Hello World!</p>
      <mm-include where="IsMenuDriven">
         <mm-group id="mobileGreeting">Hello Mobile World!</mm-group>
      </mm-include>
   </body>
</html>
```

Example 2: hello.jsp

```
<%@ taglib uri="mmJSPtaglib" prefix="mm" %>
<mm:page>
<html>
   <head>
      <title>Hello World!</title>
   </head>
   <mm:structure id="structure_1" where="IsMenuDriven">
      <mm:group-ref idref="mobileGreeting" type="normal" depth="0"
       display="all"/>
   </mm:structure>
   <body>
      <p>Hello World!</p>
      <mm:include where="IsMenuDriven">
         <mm:group id="mobileGreeting">Hello Mobile World!</mm:group>
      </mm:include>
   </body>
</html>
</mm:page>
```

Mix mm-tags and mm:tags

Using both mmXHTML and the WebLogic Mobility Server JSP tags in a single file should be avoided. For optimal processing of JSP files with WebLogic Mobility Server, you should use the JSP tag library throughout. WebLogic Mobility Server is, by default, configured to generate an error if mixed tag syntax occurs in a page that is requested from a device other than a PC browser.

In the *mis.properties* file (located in the **WEB-INF/classes** folder), `mis.transform.checkMixedTagContent` is set to “true”. Changing the value of this property to

"false"will discontinue syntax checking and will result in an increased performance overhead when processing content for a full browser and is not recommended.

Exception Handling and Diagnostics

If an exception is generated while processing the JSP tags, a `javax.servlet.jsp.JSPTagException` will be thrown. This might occur, for example, when trying to process an incorrect tag attribute. A try-catch-finally block can be added to intercept the exception; otherwise the error will be displayed in the Application Server console.

Other information can be gathered using output from the Diagnostic Console. Diagnostic messages are produced during the processing of the JSP tags and can be helpful for troubleshooting. For more information on diagnostics, see the section "Work with Diagnostics."

Optimize Performance with the JSP Tag Library

As was mentioned in the previous section, many performance gains can be realized by using the WebLogic Mobility Server JSP tag library to mobilize web content that is being requested by the full browser device class. This section explains the ways to realize the most benefit from these enhancements.

WebLogic Mobility Server Transformation Steps

There are several steps involved in the WebLogic Mobility Server transformation process. Some of these steps can by bypassed to achieve optimal performance using the WebLogic Mobility Server JSP tag library.

The typical set of steps is as follows:

- Device recognition
- Buffer set up
- Content execution (either by WebLogic Mobility Server or the tag library)
- Content check
- Content transformation
- Buffer flush (output)

The WebLogic Mobility Server JSP tag library is able to streamline the process of delivering content to a full browser device by doing much of the pre-processing itself. The required mobility tags for full browser are run and remaining tags stripped off. This allows some of the WebLogic Mobility Server transformation processing (Step 5) to be bypassed.

There are several ways to make these optimizations:

- Changing properties in the *mis.properties* file.

  (<webapp dir>\WEB-INF\classes*mis.properties***)**

- Setting attributes of the `<mm:page>` tag.
- Placing a *device.xml* file in the following folder to emulate a full browser device profile:

  WEB-INF\classes\com\mobility\util

Optimize Full Browser Performance

Here are the configuration options that can be used to optimize performance when delivering to full browser devices.

Skip Multi-Mode Tag Checking

For JSPs that are known to use only the WebLogic Mobility Server JSP tags (and not the WebLogic Mobility Server mmXHTML tags) it is possible to disable multi-mode tag checking. This allows WebLogic Mobility Server to assume JSP tag usage during processing.

Configuration

```
mis.jsptaglib.passthrough: true | false
```

Note: the default is true.

When content is being delivered to a full browser, setting this property in the *mis.properties* file to true causes the multi-mode content check and the WebLogic Mobility Server transformation to be skipped. This is advantageous for JSP files that are known to have only mm: tags marking up the content (for example, they only use the WebLogic Mobility Server JSP tags). The JSP taglib will perform the processing needed. For content being requested for devices other than a full browser, the content check and transformation is still performed by WebLogic Mobility Server.

Skip Multi-Mode Tag Checking for Specific Files in an Application

It is also possible to skip multi-mode tag checking for individual files known to use only the WebLogic Mobility Server JSP tags.

Configuration

```
mis.passthrough.patterns: /patterns/*
```

Setting this property results in significant performance benefits for the files that match the pattern specified in this the WebLogic Mobility Server property. For example, setting the pattern to *.jsp would result in all files with the .jsp extension bypassing the buffer setup, content check, and the WebLogic Mobility Server transformation when the requesting device is a full browser. This is advantageous if it is known, for example, that all .jsp pages in a web application exclusively use the WebLogic Mobility Server JSP mark-up.

Flag a Page for Optimization

Content authors can also flag a given page as only using the WebLogic Mobility Server JSP tags. WebLogic Mobility Server will process the page the first time it is accessed, and then add the URL to an internal JSP-only list. Subsequent requests will result in the same performance benefits of setting the `mis.passthrough.patterns` property above.

Configuration

```
<mm:page bypass="true">
```

Skip Device Recognition

It is possible to skip device recognition completely if a JSP will only ever be accessed from full browser devices as in the scenario where multiple app servers are used with some dedicated to PC access.

Configuration

```
mis.bypass.patterns: /patterns/*.
mis.fullbrowser.device: [Mozilla/2 | Mozilla/3 | Mozilla/4  | Mozilla/5]
```

Note: the default is Mozilla/5.

If the requested file matches the pattern in the `mis.bypass.patterns` property WebLogic Mobility Server will assume that it is delivering to the device specified in the `mis.fullbrowser.device` property. This will improve the performance by having the file processed by the JSP taglib, the device recognition step is skipped to further increase performance. The device attributes of the full browser are stored in the *device.xml* file that is included in the *mcpfilter.jar* file. However, if a file of the same name exists in the web application's **classes** directory (see the preceding section, "WebLogic Mobility Server Transformation Steps"), then this file will be used instead. This allows the user the flexibility to add device attributes.

Deliver a Web Application Without a WebLogic Mobility Server License or Filter

Configuration

```
mis.fullbrowser.device: [Mozilla/2 | Mozilla/3 | Mozilla/4  | Mozilla/5]
```

WebLogic Mobility Server-ready JSPs can be deployed on application servers that do not have a WebLogic Mobility Server filter configured. The same application can be deployed widely and, where WebLogic Mobility Server is present, will be ready for multi-channel delivery. Where WebLogic Mobility Server is not present, the web application will continue to function for full browser delivery. The WebLogic Mobility Server JSP taglib pages that are being requested by any browser will behave as if there are no WebLogic Mobility Server mark-up tags in the content - the taglib will self-process the file and the output sent to the requesting browser without being further processed by WebLogic Mobility Server. In this scenario, full browser transformed content will be delivered to any accessing device. The preceding configuration (for example Mozilla/2 | Mozilla/3) determines which full browser transformation setting is used.

Work with XHTML

Converting your existing HTML documents into XHMTL documents is one of the first steps in preparing a current web application for multi-channel use. All XHTML documents must be "well-formed". This means that they must follow XML syntax and document rules.

This section introduces the key changes you need to make to existing mark-up and how these changes can be automated.

Syntax Rules

The syntax rules require that you:

- Close all elements
- Terminate empty elements
- Quote all attribute values
- Give values to all attributes
- Define all element and attribute names in lower case (because XHTML is case-sensitive)
- Nest elements correctly

Close all Elements

Unlike HTML, all elements must have an opening and closing tag.

In HTML, this is allowed:

```
<p>This is a paragraph.
```

In XHTML, the <p> tag must be closed:

```
<p>This is a paragraph.</p>
```

Terminate Empty Elements

Some tags are termed "empty" because they have their functionality self-contained within the tag (such as a line break <br> or an image <img> tag) and do not have separate closing tags.

They do, however, need to be closed. To make these tags well-formed, add a slash (/) before the final angle bracket (>).

For example,

```
<br/>
<img src="myimg.jpg"/>
```

Quote all Attribute Values

Delimit all attributes with double quotation marks:

```
<table width="100%">
```

Supply Values for all Attributes

All attributes must have explicit values. Attribute minimization is forbidden. For example, the following HTML code has a minimized attribute "checked":

```
<input type=checkbox checked/>
```

Correct syntax requires that a Boolean attribute whose value is implicit in HTML should, in XHTML, be set equal to itself. Thus, the preceding example should be written:

```
<input type="checkbox" checked="checked"/>
```

Define all Element and Attribute Names in Lowercase

XML is case-sensitive and since XHTML DTDs define elements and attributes in lowercase, content needs to obey this requirement.

This is illegal in XHTML:

```
<H1>My Big Title</H1>
<Table width="90%">
```

Correct XHTML form:

```
<h1>My Big Title</h1>
<table width="90%">
```

Nest Elements Correctly

The following code shows two elements incorrectly nested:

```
<p>This is bold <b>text</p></b>
```

The correct nested format is:

```
<p>This is bold <b>text</b></p>
```

Encode Non-US-ASCII Characters Using URL Encoding

Non-US-ASCII characters are not valid in hrefs or any other URL attribute values (RFC 1738). This means the author must encode such characters using URL encoding. URL encoding of a character consists of a "%" symbol, followed by the two-digit hexadecimal representation (case-insensitive) of the ISO-Latin code point for the character.

Note: Different web servers use different encodings.

Document Rules

XHTML is HTML defined as an XML application. The XML document rules require that all documents have one root element and conform to the XML specification. Any XHTML documents that you work with must also follow this convention.

A Root Element is Required

The root element contains all the other elements on a page. In XHTML, the root element is the `<html>` element.

XML Declaration Required

The XML declaration declares that the current document conforms to the XML specification. The declaration has three attributes: version, encoding, and standalone.

The shortened syntax is as follows:

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

Note: There is no space separating the "?" from the angle brackets.

Automate HTML to XHTML Mark-up

HTML Tidy is a popular tool for converting HTML documents into clean XHMTL documents. Developed by Dave Raggett, it is available on CD2 or as a free download from:

http://www.w3.org/People/Raggett/Tidy

Some of the features included in this tool are:

- detection and correction of mismatched tags
- addition of quotation marks to attribute values
- correction of incorrectly nested elements
- location of misplaced elements
- conversion of element and attribute names to lowercase

Enable Less Strict Document Checking

Enabling less strict XML parsing for portals/3rd party integration is often necessary. In order to facilitate this, the customized XML parser can read documents that are to be mobilized by WebLogic Mobility Server.

The customized XML parser has the following characteristics:

- High performance when dealing with XHTML, outperforming market leading standard parsers, such as Xerces
- XML/XHTML processing capability
- Support for "relaxed" parsing of non-XML compliant documents; a feature that is vital for integration with many portals and third party products

For consistency with XHTML standards, this parser is configured by default to perform strict parsing (for example to reject stand-alone "&" symbols in XHTML attributes). For integration with pre-existing content and frameworks, this strictness can be switched off by setting the Boolean

Note: If the value of this property is changed, WebLogic Mobility Server must be restarted for the change to take effect.

```
xsp.strictAttribute: true
```

When the `xsp.strictAttribute` property is set to true or if the property is not present, the enforcement of strict XML encoding standards leads to parsing errors occurring when less strict XML is encountered in attributes of URLs. For example, if the following URL was encountered,

```
<a href="http://www.whereami.com/index.jsp?location=Dublin&Street=OConnell"/>
```

then a parsing exception would be thrown due to the presence of the & in the URL. The XSP parser expects an entity reference (that is, `&`) or a character code reference (`that is, &#141 or &#x141`) and when it finds just the "`&Street`", it assumes that it has found a malformed entity

```
xsp.strictAttribute: false
```

To accommodate situations such as the preceding one, the syntax checking / entity rewriting of the XML parser needs to be liberal, that is, the "malformed entity" checking in the XML parser would need to be disabled (set to false). Setting the `xsp.strictAttribute` to false disables this exception handling and allows processing of "**`non-strict`**" XML content.

When the property is set to `xsp.strictAttribute: false` the WebLogic Mobility Server parser provides more flexibility for dealing with the "&" character within attributes, by allowing for non-strict XML to be parsed. This is to minimize the impact on portal / application integration where the content of attributes may not be strictly legal XML.

Example

In the following sample code, the line `<a href …></a>` contains non-strict XML. When `xsp.strictAttribute` is set to false, WebLogic Mobility Server, as shown in the following graphic, processes the output from this code. When `xsp.strictAttribute` is set to true, however, WebLogic Mobility Server generates an exception while attempting to process this code.

```
<mm-body id="bd_company_details" idref="hd_company_details">
 <mm-img src="ma_logo.wbmp" where="ImgWBMPSupported"
 <p>ABC Company Ltd.</p>
 <p>North Business Park, Circular Road, Dublin, Ireland</p>
 <a href="More Info?a=b&x=y">More Info</a>
 <hr> <p>Phone: 888-000-111</p> <hr>
 <p>Fax: 000-888-111</p>
</mm-body>
```

WebLogic Mobility Server will overlook some less well-formed content if `xsp.strictAttribute` is set to false.

Use of Non-Strict XML

Organize Content

This section introduces a key concept that forms the foundation for organizing and managing your content – groups.

The section introduces the following mmXHTML tags:

- `<mm:group>`
- `<mm:head>`
- `<mm:body>`
- `<mm:id-ref>`

Introducing Groups

Groups are a way of organizing your content for the purpose of having greater control over what gets displayed on each device. Grouping involves dividing your content into logical sections and assigning these sections an ID that can be referenced when deciding which groups of content should be delivered to specific devices.

WebLogic Mobility Server introduces the notion of implicit and explicit groups. Implicit groups are created by assigning an `id` to an existing XHTML tag like a form or a table.

Explicit groups are those you create with the WebLogic Mobility Server `<mm:group>` tag. This element enables you to split content into a number of separate components that can be referenced individually, for example, organizing a news page into national, regional and local news groups. Alternatively, you can use `<mm:group>` to collect a number of resources (such as a table and a form) and reference them as a single unit.

Create Explicit Groups

The `<mm:group>` tag has a number of attributes to identify and tailor the content group and allow it to be referenced by other mark-up tags.

Define a Named Group

Use the following tag to define and assign an id to a group:

```
<mm:group id="..." title="...">
```

where `id` is a unique identifier for your group and `title` is the title of the group.

When assigning an `id` name you should select a consistent naming convention to make it easier to manage your groups. Choosing a meaningful name will also remind you of the group's purpose.

Note: The group `id` must not contain spaces.

After the `<mm:group>` tag, you can assign a heading and define the body. Complete the group with the closing tag:

```
</mm:group>
```

For example,

```
<mm:group id="gp_details" title="About Us">
  …
</mm:group>
```

A group may contain a single head element (optionally) and zero or more body elements. The body element is linked to the `id` of the head element. If no head is present the first body element must link to the `id` of the group element.

```
<mm:group>
    <mm:head>…</mm:head>
    <mm:body>   (Note: must be linked to id of mm:head)
    </mm:body>
</mm:group>
```

Assign a Heading to a Group

A heading can be used to facilitate navigation on menu-driven devices. Smaller screen devices often split content into smaller pieces to accommodate the inherent memory and bandwidth restrictions. A heading is sometimes needed to give context to the piece of content that is being displayed. The `<mm:head>` tag is used for this purpose.

The heading can be used in another way to improve navigation on menu-driven devices. If the *mis.properties* file is configured to enable “Back To Top” functionality, the heading inside the `<mm:head>` tag will form a link at the end of split pages which will allow the user to return to the top of the group based on the hierarchy of the current document.

For example, if you grouped an article about elephants and gave it a heading “Elephants”, each of the subsequent split pages will have a link at the end that would read “Back to Elephants >>“.

Note: If no heading is included, the `title` attribute is used both in the navigation menu and as the “Back To Top” text.

If a heading is assigned to a group, it may be part of the existing content, or may be explicitly added for menu-driven navigation purposes.

Use the following tag to assign a heading to a group:

```
<mm:head id="name" useradded="yes|no">
```

where `id` is the unique identifier for your heading and `useradded` indicates whether this heading is already part of the content (`useradded="no"`) or has been explicitly added (`useradded="yes"`). The default is (`useradded="no"`).

Place the content of the heading after this tag, including any formatting that you require and complete it with the closing tag:

```
</mm:head>
```

For example,

```
<mm-:head id="hd_details" useradded="no" > <b>Details</b></mm:head >
```

Define the Body Content for a Group

The content for a group follows the group heading. The following tag can be used to define the content for the group:

```
<mm:body id="..."  idref="....">
```

where `id` is the unique identifier you give your body and `idref` is the identifier of the heading associated this body content.

For example,

```
<mm:body id="bd_address" idref="hd_details">
```

The first body in a group is linked to the group's heading with the `idref` attribute. If a group has more than one body, each subsequent body is linked to the previous one. If a group has no heading, the first body should be linked to the group's `id`**.**

Follow the opening tag with the content and complete it with the closing tag:

```
</mm:body>
```

You create the content for the group just like you would for any normal web page: XHTML content (such as tables or general formatting), JavaScript, JSP code, text and images can be placed within the group tags.

The code example on the next page illustrates how the contact details for an organization (name, address, phone and fax numbers) could be defined as a group.

Place Content Outside a Group's Body

Text inside a group, but outside both the group head and body will not appear on menu-driven devices that are using a structure to reference the group. The following code snippet uses a structure to reference a group called "group1". This group contains two sentences: one inside the body; one outside. For menu-driven devices, only the text inside the body is delivered to the device.

```
...
<mm:structure id="str1" where="IsMenuDriven">
    <mm:group-ref idref="group1" depth="flat" type="normal" display="all"/>
</mm:structure>
...
<mm:group id="group1" title="My Group1">
  <mm:head id="head1" useradded="yes">Heading Text<br/></mm:head>
  <p>This text will NOT appear on menu-driven devices, but will appear on full
browsers.</p>
  <mm:body id="body1" idref="head1">
    <p>This text will appear on menu-driven devices AND full browsers.</p>
  </mm:body>
</mm:group>
...
```

This feature allows content authors to specify only parts of a large section of content for delivery to menu-driven devices by using multiple bodies within a single group.

Examples

mm-group

The following examples illustrate the different ways in which `<mm-group>` can be defined.

Example 1: mm-group with mm-head and mm-body

This example demonstrates the basic behavior of `<mm-group>` with `<mm-head>` and `<mm-body>`**.** Notice that the heading "Company Details" is in bold on the menu-driven device. When `useradded` is set to yes, "Company Details" does not appear on full browser devices.

<mm-group id="gp_company_details" title="Details">

```
    <mm-head id="hd_company_details" useradded="yes">
            Company Details
    </mm-head>
    <mm-body id="bd_company_details" idref="hd_company_details">
```

```
        <p>ABC Company Ltd.</p>
        <p>North Business Park, Circular Road, Dublin, Ireland</p>
        <p>Phone: 888-000-111</p>
        <p>Fax: 000-888-111</p>
    </mm-body>
</mm-group>
```

Using a group with a head and a body.

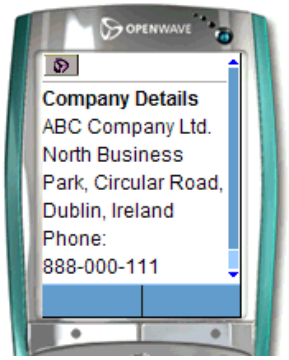


Image Courtesy of Openwave Systems Inc

Example 2: mm-group with no mm-head or mm-body

Here, neither `<mm-head>` nor `<mm-body>` are used inside the group. The output is essentially the same as above, except that as no heading has been defined, all the text is displayed in the same font.

```
<mm-group id="gp_company_details" title="Details">
    Company Details
    <p>ABC Company Ltd.</p>
    <p>North Business Park, Circular Road, Dublin, Ireland</p>
    <p>Phone: 888-000-111</p>
    <p>Fax: 000-888-111</p>
</mm-group>
```

Using mm-group with No Head or Body

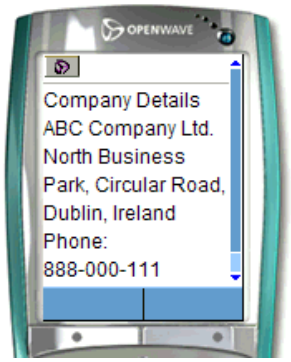


Image Courtesy of Openwave Systems Inc

Example 3: mm-group with an mm-head but no mm-body

This example shows the behavior of an `<mm-group>` with an `<mm-head>` but no defined `<mm-body>` tag. Because the heading has been defined, it is differentiated from the rest of the text when seen on a menu-driven device.

```
<mm-group id="gp_company_details" title="Details>
    <mm-head id="hd_company_details" useradded="no">
            Company Details
    </mm-head>
        <p>ABC Company Ltd.</p>
        <p>North Business Park, Circular Road, Dublin, Ireland</p>
        <p>Phone: 888-000-111</p>
        <p>Fax: 000-888-111</p>
    </mm-group>
```

mm-group with mm-head and no mm-body

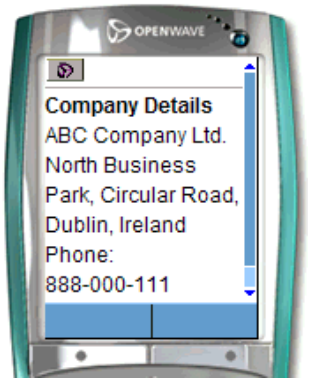


Image Courtesy of Openwave Systems Inc

Create Sub-Groups

As you organize your content, on certain occasions you may want to create sub-groups; in other words, create groups within groups. Defining a group within the body of the parent group creates sub-groups.

The following example demonstrates how the company details group is further divided into two sub-groups: one for the name; the other for the contact information.

```
<mm-group id="gp_company_details title="Details">
   <mm-head id="hd_company_details" useradded="no">
      Company Details
   </mm-head>
   <mm-body id="body_company_details" idref="hd_company_details">
      <mm-group id="company_name" title="Name">
         <mm-head id="hd_company_name" useradded="no">
            Company Name
         </mm-head>
     <mm-body id="bd_company_name" idref="hd_company_name">
            <p>ABC Company Ltd.</p>
         </mm-body>
      <mm-group>
      <mm-group id="contact_info" title="Contact">
         <mm-head id="hd_contact_info" useradded="no">
            Contact Information
         </mm-head>
         <mm-body id="bd_ contact_info" idref="hd_contact_info">
            <p>North Business Park, Circular Road, Dublin, Ireland</p>
            <p>Phone: 888-000-111</p>
            <p>Fax: 000-888-111</p>
         </mm-body>
      </mm-group>
  </mm-body>
</mm-group>
```

It is important to note that when using nested groups (sub-groups), text that displays outside the head and body of a sub-group will not appear when transformed for menu-driven devices even though it may, in fact, be inside the body of an outer group. The reason for this is that the text is interpreted by WebLogic Mobility Server as being logically part of the sub-group rather then as text within the body of the surrounding group.

Work with Implicit Groups

Implicit groups are XHTML elements that can be assigned an `id` attribute, such as a table, a form or a text area. Rather than re-marking these with the `<mm-group>` tag, you can reference them directly within a layout through their existing `id` attribute. For other content that does not normally have an explicit ID (such as a paragraph, or a sentence) you can assign one by wrapping it in a `<div>` or `<span>` tag as follows.

```
<div id="ceo_1" align="right">John Doe</div>
```

Note: Implicit groups can only be referenced from within layout files. Using an `<mm-id-ref>` does this. Implicit groups are not allowed for menu-driven devices (that is, groups that are referenced inside an `<mm-structure>` tag).

Reference Implicit Groups

Use the following tag to reference an implicit group:

```
<mm-id-ref id="content_id">
```

where `content_id` is the ID assigned to the content.

Target Content for Specific Devices

Groups and layouts provide a lot of flexibility for customizing the content being delivered to mobile devices

As you organize your content, you may decide that you want to only use part of the material within a group. For example, you may have an introductory page with ten paragraphs but you only want the first, third and final paragraph to be delivered to mobile devices. Alternatively, you might have a table, from which you want to extract content from specific non-adjacent cells for the smaller screen devices.

There are two methods for managing such content:

- Define content to be included or excluded when a particular device is being targeted.
- Link the content together through a series of `<mm-body>` elements.

Link Content Together

Since a group's content can be spread across a number of locations (such as in different cells in a table) you need to wrap each element in its own body definition.

To maintain the order in which the content gets displayed, you effectively create a linked list with the body tags: the first body tag references the heading tag, and each subsequent body tag references the `id` attribute of the body tag preceding it.

```
<mm-group id="introduction" grouptitle="Introduction">
    <mm-head id="hd_introduction" useradded="no">Introduction</mm-head>
        <!-- Note the first body is linked to the heading -->
        <mm-body id="bd_1" idref="hd_introduction">
            <p>This is paragraph 1</p>
        </mm-body>
        <p>This is paragraph 2</p>
        <!-- Note the 3rd paragraph is linked to the 1st paragraph -->
        <mm-body id="bd_3" idref="bd_1">
            <p>This is paragraph 3</p>
        </mm-body>
        <p>This is paragraph 4</p>
        <p>This is paragraph 5</p>
      <!-- Note the 6th paragraph is linked to the 3rd paragraph -->
        <mm-body id="bd_6" idref="bd_3">
            <p>This is paragraph 6</p>
        </mm-body>
</mm-group>
```

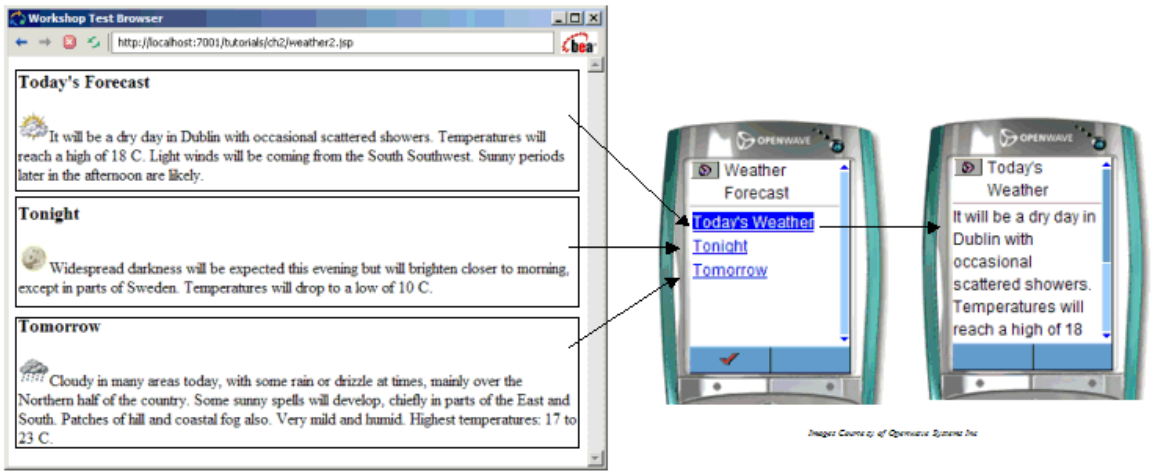
This technique is also useful if you want to extract specific content from a table. Wrap the table in a group and then, for each cell that you want to extract, wrap its contents within `<mm-body>`. Again, the order in which you link definitions together will determine the order in which they are displayed. You could, for example, have the content of the last cell appear before the contents of the middle cell.

Reference Groups

Grouping allows you to use arrange the same source content into different layouts that best suit the dissimilar types of requesting devices.

The following graphic illustrates how the groups, once defined, can be represented for display on a handheld device.

Using Groups to Create Content for Mobile Devices



Create Header and Footer Groups for Menu-driven Devices

This section explains how to display a header and footer on each page presented on a menu-driven mobile device. It refers specifically to the following mmXHTML tags:

- `<mm-group>`
- `<mm-structure>`
- `<mm-include>`

And the following attributes (within the `<mm-structure>` tag)

- `idheader`
- `idfooter`

Create header and footer groups

Insert the pieces of content that you want to display as header and footer into the page. Then define header and footer groups by inserting `<mm-group>` tags around each piece of content and assigning it a group ID.

Note

- You must use `<mm-include>` tags to ensure that the content of these groups is only included for menu-driven devices
- To ensure consistent spacing within header and footer groups, use the `<span>` tags to enclose the header and footer content

In the following extract, for example, a `<mm-group>` tag has been inserted around the footer content “Copyright”, and the group has been assigned a group ID of “footer”.

```
<mm-group id="footer" title="Example Footer">
<mm-include where="IsMenuDriven">
       <span>
       ----------------<br/>
       <b>Copyright</b><br/>
       ----------------<br/>
       </span>
</mm-include>
</mm-group>
```

Set the idheader and idfooter Attributes in the <mm-structure> Tag

You will then need to modify the <mm-structure> tag to specify the mm-groups you want to use as a header and footer. This involves setting the idheader and idfooter attributes to point to the group IDs of the mm-groups to be used. The following example illustrates how to do this:

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//MOBILEAWARE//DTD MMXHTML 1.2//EN"
"http://www.mobileaware.com/DTD/mmxhtml_1.2.dtd">
<html>
       <head>
              <title>Header Footer Example</title>
       </head>
       <mm-structure id="mm-structure-1" where="IsMenuDriven"
idheader="header" idfooter="footer">
              <mm-group-ref idref="main-group" type="normal" depth="0"
display="all"/>
       </mm-structure>
       <body>
              <mm-group id="header" title="Example Header">
              <mm-include where="IsMenuDriven">
       <span>
                     -----------------<br/>
                     <b>Sports Page</b><br/>
                     -----------------<br/>
                     </span>
</mm-include>
              </mm-group>
              <mm-group id="main-group" title="Main">
                     Uefa says Manchester United's Champions League
clash with Juventus will go ahead.
              </mm-group>
              <mm-group id="footer" title="Example Footer">
<mm-include where="IsMenuDriven">
                     <span>
                     -----------------<br/>
                     <b>Copyright</b><br/>
                     -----------------<br/>
                     </span>
</mm-include>
              </mm-group>
       </body>
</html>
```

Notes

- You can use the same mm-group as both a header and a footer, if required
- A page can have a header, a footer, both, or neither
- You can place the header and footer groups anywhere on the page
- If more than one mm-structure exists for the same class of device on a page, ensure that the idheader and idfooter attributes exist on the first of these mm-structures

Create Conditional Content

This section describes the creation of conditional content through the use of the following syntax:

- The `<mm-include>` element
- The `<mm-exclude>` element
- The "`where`" attribute

Create Conditional Content

By default, everything defined within the body of a group is displayed when you reference the group, but occasionally you will want finer control when specific devices are being targeted.

For example, you might want line breaks (`<br/>`) inserted into an address group when it displays on a mobile phone, so that the address wraps neatly. Alternatively, you might decide to exclude the address altogether and just provide the phone number.

Another example might be changing the amount of introductory text you provide on a page. You could display ten paragraphs of text on a desktop browser but exclude the last five paragraphs when it is displayed on a PDA.

Include or Exclude Content

WebLogic Mobility Server provides two tags to include or exclude content: `<mm-include>` and `<mm-exclude>`.

Wrap the content you want included or excluded in these tags. Both tags require the `where` attribute. The `where` attribute allows conditional expressions to be built up to match specified Device Repository attributes.

The following examples demonstrate the usage of these tags. The example *Including Line Breaks* shows how to include line breaks within an address group when a specific device is targeted. The example "Excluding an Address for a Class of Device" demonstrates excluding text.

Example: Including Line Breaks

```
<mm-group id="company_details">
    <mm-head id="hd_company_details" useradded="no">Company Details</mm-head>
    <mm-body id="bd_company_details" idref="hd_company_details">
        <p>ABC Company Ltd.</p>
        <p>North Business Park
        <mm-include where="IsMenuDriven"><br/></mm-include>
        Circular Road
        Dublin
        Ireland
        </p>
        <p>Phone: 888-000-111</p>
        <p>Fax: 000-888-111</p>
    </mm-body>
</mm-group>
```

Example: Exclude an address for a class of device

```
<mm-group id="company_details">
    <mm-head id="hd_company_details" useradded="no">Company Details</mm-head>
    <p>ABC Company Ltd.</p>
    <mm-exclude where="IsMenuDriven">
        <p>North Business Park, Circular Road,Dublin,Ireland </p>
    </mm-exclude >
    <p>Phone: 888-000-111</p>
    <p>Fax: 000-888-111</p>
</mm-group>
```

The “where” Attribute

As can be seen from the preceding examples, the `where` attribute can be used to control the output based on device type. It also can identify devices based on specific attributes.

The `where` attribute can be used with the following tags:

- `<mm-include>`
- `<mm-exclude>`
- `<mm-img>`
- `<mm-structure>`
- `<mm-layout>`
- `<mm-table-model>`

Usage of the “where” Attribute

The conditional expressions following the `where` attribute are evaluated based on values of the referenced device profile attributes. If the condition evaluates to true, then the content enclosed by the tag will be added/excluded/manipulated for that device (depending on the tag’s functionality).

If the `where` attribute does not evaluate to true, it will be ignored (except when `<mm-exclude>` is used). For example:

```
where="AttributeName==AttributeValue or AttributeName2==AttributeValue2"
```

Note: The “where” attribute quoted string must not contain line breaks.

The `where` attribute:

- Is compatible with the scripting language, Python.
- Provides support for conditional logic: and, or and not.
- Provides support for comparison operators: ==, <>, >, <, >= and <=.
- Provides support for comparison expressions with non-literal values. These non-literal values must be attributes of the device defined in the Device Repository.
- Provides support for nested expressions using parentheses.
- Provides support for partial string matching (for example startswith, endswith, find).
- Provides user-friendly compilation warnings by throwing detailed exceptions.

- Provides support for select(<condition>,trueValue,falseValue), for example select(UsableWidthPixels>200,"BIG","SMALL") will return "BIG" if the expression is true and "SMALL" if the expression if false.
- Multiple "==" expressions can be combined with a logical AND or logical OR operator by adding an "and" or "or" (lowercase) between expressions.
- Within the "==" expressions, number and Boolean values can be entered as is, but strings must be surrounded by single quotes.

Here is an example using Boolean, string and number comparisons:

```
<mm-include where="IsMenuDriven and
UAProf.SoftwarePlatform.OSName=='Microsoft Windows' and
UsableHeightPixels < 250">
       ...Some content...
</mm-include>
```

Note: When referring to User Agent Profile (UAProf) attributes in the database, the full prefix is required (for example `UAProf.SoftwarePlatform.OSName`, `UAProf.BrowserUA.BrowserName`).

Proper "where" Usage

Valid expressions when evaluated will cause the `where` clause to evaluate to either true or false. Examples of valid expressions are highlighted in the following sections.

Comparison Operators

Comparison operators that are supported include: ==,<>,<,>,>= and <=.

All of the following `where` expressions are valid:

- where="UAProf.HardwarePlatform.NumberOfSoftKeys == 0"
- where="UsableWidthPixels <> 200"
- where="UsableWidthPixels < 200"
- where="UsableWidthPixels > 200"
- where="UsableWidthPixels >= 200"
- where="UsableWidthPixels <= 200"
- where="MLVersion == 'WML1.1'"

Note: When comparing against strings the literal string values must be surrounded by single quotes.

Conditional Logic

The logic operators supported include "and", "or", and "not".

All of the following `where` expressions are valid:

- where="UsableWidthPixels < 200 and UsableHeightPixels < 300"
- where="UsableWidthPixels < 200 or UsableHeightPixels < 200"
- where="not IsPortraitPDA"
- where="not MP3Supported"

- where="MP3Supported"
- where="UsableWidthPixels < 200 and not IsPortraitPDA"

Non-literal Comparisons

Device attributes can be compared against other device attributes as follows:

- where="UsableWidthPixels == UsableHeightPixels"

Nested Expressions

Sub expressions within the `where` expression can be nested using parentheses.

The following are valid:

- where="UsableWidthPixels < 200 and UAProf.BrowserUA.TablesCapable) or IsPortraitPDA"
- where="((UsableWidthPixels < 200 or IsPortraitPDA) and UsableHeightPixels < 300)"
- where="not (UsableWidthPixels < 200 and UsableHeightPixels < 300)"

Spacing

Spacing between comparison operators will NOT invalidate the expression.

These are both valid expressions.

- where="UsableWidthPixels < 200"
- where="UsableWidthPixels<200"

Partial Matching

The following are all valid.

- where="DeviceName.endswith('PAQ')"
- where="DeviceName.startswith('IP')"
- where="ImgGIFSupported"
  (often used with the `<mm-img>` tag)

Invalid Expressions

There are a number of reasons why a `where` expression may be invalid.

In Development Mode, if there is an invalid `where` expression then a warning is sent to the WebLogic Mobility Server Application Server console. If there is more than one reason why an expression is invalid, only the first reason will be reported.

In Production Mode, the expression evaluates to false but no warning is sent to the console.

Incorrect Syntax

In these instances, the expression is not constructed properly. The expression is not built according to the correct syntax rules.

- where="UsableWidthPixels"
  Missing comparison.

- where="UsableWidthPixels 200"
  Missing comparison operator.
- where="UsableWidthPixels >= 200 and"
  Missing clause.
- where="(UsableWidthPixels < 200 or UAProf.HardwarePlatform.ColorCapable"
  Missing parenthesis.
- where="UsableWidthPixels < 200and UsableHeightPixels < 300"
  Unrecognizable keyword "200and".

Use Device Attributes Containing Hyphens

Some attributes have a hyphen as part of their name (for example UAProf.PushCharacteristics.Push-MsgSize). Because the where clause is evaluated as a Python expression, the hyphen is interpreted as a minus symbol, so if you are using one of these attributes in a where condition, you **must** replace the hyphen with an underscore.

for example where = UAProf.PushCharateristics.Push_MsgSize>0

Unrecognizable Identifier Names

The following are well-constructed expressions but the device attribute does not exist within the WebLogic Mobility Server system.

Note: In the second example, because of the missing single quotes, WebLogic Mobility Server assumes that DeviceName is being compared to another device attribute. This will result in an error.

- where="UseableWidthPixels == 200"
  UsableWidthPixels is misspelled
- where="UAProf.HardwarePlatform.Vendor==Ericsson"
  No single quotes around "Ericsson"

Type Mismatches

Type mismatches occur when the device attribute type does not match the type of the value or device attribute against which it is being compared.

Mismatches follow the rules specified by the Python language for type errors. For example, the following three examples will produce a type mismatch.

- where="UsableWidthPixels"
  UsableWidthPixels is not Boolean.
- where="UsableWidthPixels == ‘wide’"
  UsableWidthPixels is not a string.
- where="UsableWidthPixels == 100+’640’"
  Cannot add an integer and a string value.

The following example will NOT produce a type mismatch

- where = "‘a’ > 7000"
  The letter "a" evaluates to its ASCII value of 97 before being compared to 7000.

Miscellaneous Issues

- Division by 0 will simply return false. No exception will be thrown
- If an attribute is valid but is not an attribute of the device in question, a naming exception is thrown
- Java method calls will not be evaluated. A naming exception will be thrown
- Short circuit logic applies. This means that if a syntax error has not been encountered before the outcome of the statement has been determined, there will be no error. WebLogic Mobility Server does not continue to evaluate a statement once a result has been established

Use “where” with Request Headers

In addition to using “where” expressions with Device Repository attributes, it is possible to use “where” expressions with information from Request Headers.

This is done using the `getHeader("headerName")` function in a “where” expression. The "headername" is case-insensitive and allows any string.

Example

To include text if the Accept Header contains "gif", you could use the expression:

```
<mm-include where="find('gif',getHeader('accept'))'>Found Gif in accept
header</mm-include>
```

Note: getHeader returns "None" (the python version of null, not the string) if the requested header is not present.

Alternatively, `getHeaderWithDefault('headerName','valueIfNotPresent')` can be used. This function will return the defined "valueIfNotPresent" if there is no header called "headername" for this request.

Examples of “where” Usage

Example 1: Use of “where” with “and” Operator

Here’s an example using a compound `where` condition that uses the “`and`” operator.

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//MOBILEAWARE//DTD MMXHTML 1.2//EN"
"http://www.mobileaware.com/DTD/mmxhtml_1.2.dtd">
<html>
<head>
   <title>Use of "and"</title>
</head>
<mm-structure id="include-and1" where="IsMenuDriven">
   <mm-group-ref idref="message-include-and" type="normal" depth="flat"
    display="all"/>
</mm-structure>
<body>
   <mm-group id="message-include-and" title="Message-include-and">
      <mm-head id="message-head-include-and" useradded="yes">
       MENU-DRIVEN DEVICE
      </mm-head>
      <mm-body idref="message-head-include-and" id="message-body-include-and">
        <h1>Example 1</h1>
        <mm-include where="DeviceName=='M3Gate0.6'and UsableWidthPixels ==120">
```

```
            <br/><p>This text will be included in devices named M3Gate 0.6 and
            with a viewable screen width equal to 120. </p>
          </mm-include>
          <hr/>
        </mm-body>
    </mm-group>
</body></html>
```

If the where condition returns true, a line of text is included for devices that match a specific device name AND have a `UsableWidthPixels` attribute of 120.

This body text will not appear on a PC browser.

Example 2: Use of “and not”

If the `<mm-include>` in Example 1 is replaced with the following line:

```
<mm-include where="UsableWidthPixels < 200 and not IsFullBrowser">
```

The same result will be seen in the M3Gate and Mozilla/4 browsers as in Example 1 as the text is included for **UsableWidthPixels < 200** and is not included for **“IsFullBrowser”**.

Example 3: Use of “or”

If the `<mm-include>` section in Example 1 is replaced with the following:

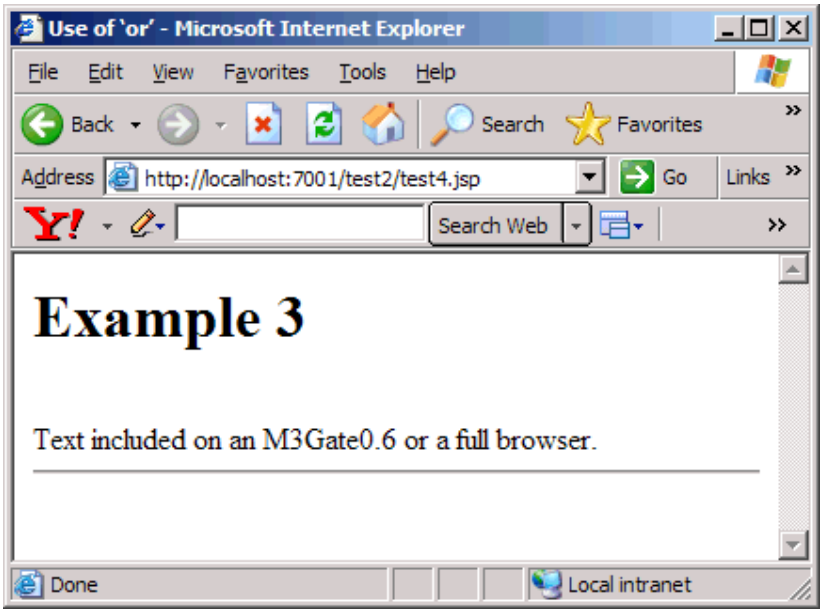
```
<h1>Example 3</h1>
<mm-include where="DeviceName=='M3Gate0.6' or IsFullBrowser">
   <br/> Text included on an M3Gate0.6 or a full browser. <br/>
</mm-include>
```

And the title tag at the top of the file is changed to:

```
<title>Use of "or"</title>
```

The include text is output on both the M3Gate and full browsers as follows:

Example 3 on a full browser and (menu-driven) device



Example 3 Result on a WML Menu-Driven Device

Example 4: Use of “endswith”

If the <mm-include> section in Example 1 is replaced with the following:

```
<h1>Example 4</h1>
<mm-include where="DeviceName.endswith('4')>0">
<br/> Text included for any device name ending with a 4.<br/>
</mm-include>
```

and the title tag is changed to:

```
<title>Use of "endswith"</title>
```

then the text inside the <mm-include> tags would be displayed on the Mozilla/4 browser because it ends with 4.

Results of Example 4 in PC browser (Mozilla/4)

Part IV Presentation of Mobile Content

Organize Content for Handheld Devices

The simple translation of the WebLogic Mobility Server mark-up into a device-specific mark-up is not the only thing that is required to develop professional looking web pages for mobile devices. It is important to remember that in order for a web page to look good on handheld devices, the content also needs to be rearranged to suit the size, shape and memory capabilities of each device. Furthermore, it is necessary to take into consideration whether the device supports mouse, stylus or keypad navigation. And finally, the appearance of the web page may be improved by the layering of styles, colors and borders onto the transformed content. WebLogic Mobility Server gives you the tools to simplify these tasks.

This section covers in detail how to arrange and style web content for presentation on the smaller handheld devices using groups. It explains how WebLogic Mobility Server transforms a document marked up with the WebLogic Mobility Server mobility tags into the mark-up of the requesting device and introduces the concept of developing a navigational structure for the smaller screen devices. Lastly, it demonstrates how the main categories of devices are differentiated and how the order of layout references in content is important in determining which layout is selected for a device.

Web Site Arranged for PDA Devices

Arrange and Style Content

As has been described earlier, WebLogic Mobility Server transforms content marked up with the mobility tags into a suitable language for the targeted device, whether it is WML, XHTML-MP, a particular version of HTML, or another format.

However, to deliver content that has simply been translated into the appropriate mark-up for the requesting device would not necessarily produce optimum results**.**

Web Site Arranged for WML Device

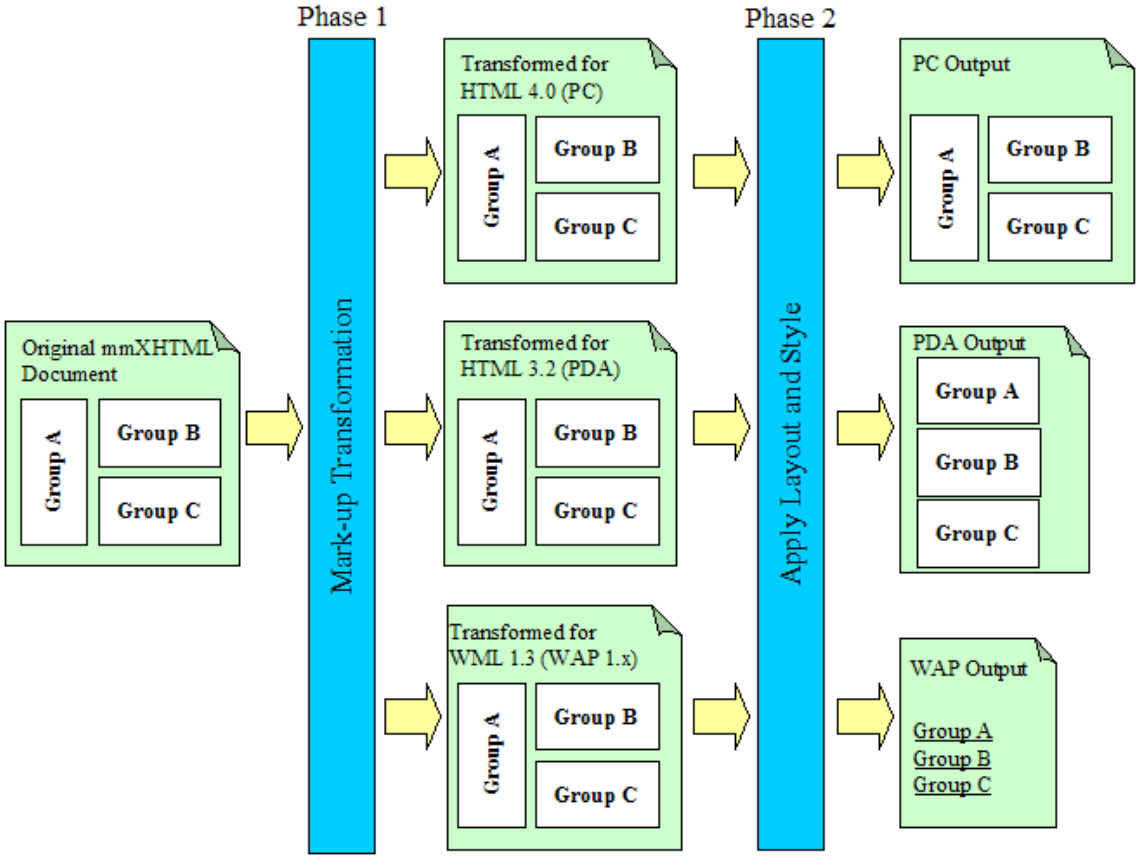
Image Courtesy of Openwave Systems Inc

Web Site Arranged for XHTML-MP Device

To address this, the transformation process is divided into two distinct phases:

- Transformation to target mark-up: (for example, transforming mmXHTML into WML 1.2) as described in earlier sections. This process is completely rule-based and requires no input from content developers.
- Application of layout and styling preferences: This process empowers content developers to create stencils describing how the output will be arranged on each of the different classes of devices. The following graphic presents the concept graphically.

The 2 Phases of Content Transformation



After the first "transformation" phase, the content has been distilled to suit the target device. Also notice in this simple example that formatting directives such as bold and italic have been removed from the WML output. After the second "layout and style" phase, the content has been restructured to suit the size and shape of the device. Typically, PDA content is arranged sequentially, WAP content is arranged as a set of menu items (or links) and PC content is presented very similarly to the original mark-up. During this second phase, WebLogic Mobility Server also performs the application of cascading style sheets for devices that cannot support style sheets locally. Style sheets are discussed in greater detail in the section “Working with Style Sheets”.

In normal circumstances, a content developer will only develop two distinct layouts for the web application; one for targeting content for PDAs and one for menu-driven devices such as WAP phones.

The next two sections will discuss in detail the considerations that have to be taken into account when restyling web content for PDAs and phones.

Restyle for PDAs

WebLogic Mobility Server provides 2 methods for authors to restyle their web content for PDAs. Which method you select depends on the type of content you are presenting.

Method 1 is used primarily for smaller documents, which can have the content groups rearranged one above the other. Method 2 works well with larger documents that benefit from being broken into several smaller pages that can be navigated through using menu links created by WebLogic Mobility Server.

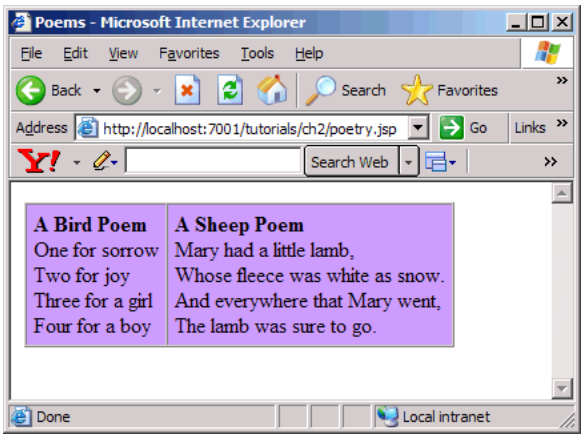
Method 1: Use layout files for PDA display

PDAs, because of their smaller screen size, often need a simpler, cleaner layout than would be used for a desktop browser. For this reason, WebLogic Mobility Server introduced the concept of layout files.

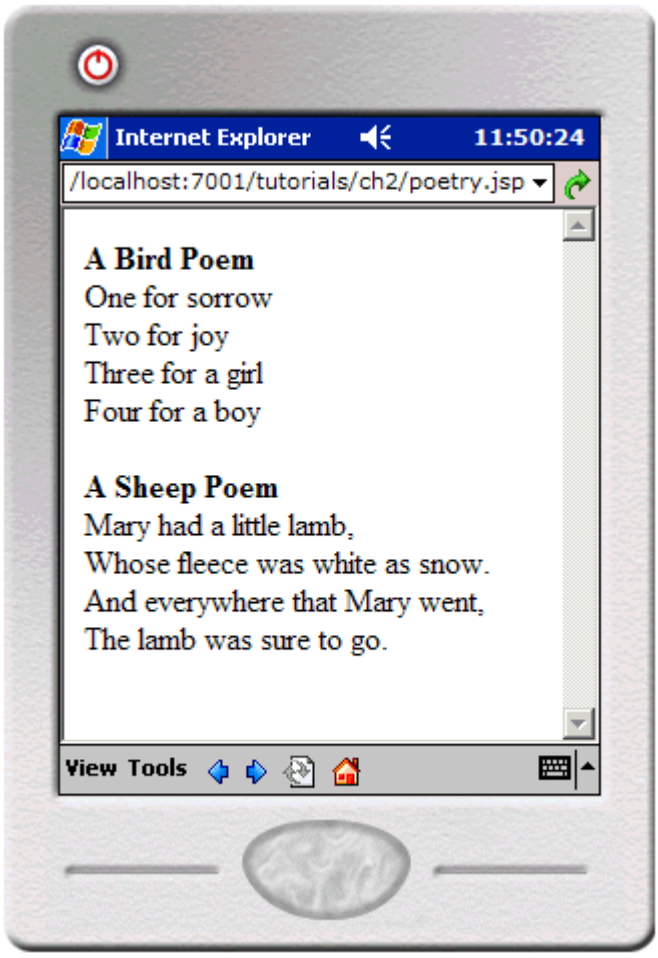
Layout files are basically skeleton template files which hold groups of content, but which are laid out in a way that is better suited to the smaller device. They closely resemble the concept of the HTML frameset that references the HTML pages that are to appear within each frame. The sections of the content, which have been identified as “groups”, can be referenced from within the specialized layout files to create a more appropriate display on the smaller device.

Content is often organized differently on PCs and PDAs as illustrated:

Content Organized for PCs



Content Organized for PDAs



In this example, the two poems can appear side-by-side in a full browser, but in order to avoid excessive wrapping, they are stacked one on top of the other when delivered to a PDA device. The following code blocks contain an example of how this is done. The `<mm:layout>` tag indicates that a layout file is to be used when the requesting device is a PDA.

Note: Layout files should contain mark-up organized for menu-driven devices and PDAs only. Setting the attribute where="IsFullBrowser" is not allowed in the `<mm:layout>` tag.

The particular layout file is identified by the `src` attribute. The content in the main file (*myFile.jsp*) has been grouped. The content of the first table cell is grouped and labelled “group1”. The content of the second table cell is grouped and labelled “group2”. These groups are then referenced from the layout file using the `<mm_id-ref>` tag.

myFile.jsp

```
<%@ taglib uri="mmJSPtaglib" prefix="mm"%>
<mm:page content="false">
<html>
<head>
<title>Restyling for PDAs</title>
   <mm:layout where="IsPDA" src="pda_layout.jsp"/>
</head>
<body>
   <table>
      <tr>
         <td><mm:group id="group1" title="Group1">
             <p>Content of group 1.</p>
             </mm:group>
         </td>
         <td><mm:group id="group2" title="Group2">
             <p>Content of group 2.</p>
             </mm:group>
         </td>
      </tr>
   </table>
</body>
</html>
</mm:page>
```

pda_layout.jsp

```
<%@ taglib uri="mmJSPtaglib" prefix="mm"%>
<mm:page>
<html>
<head>
   <title>PDA Layout</title>
</head>
<body>
   <p><mm:id-ref idref="group1"/></p>
   <p><mm:id-ref idref="group2"/></p>
</body>
</html>
</mm:page>
```

pda_layout.jsp takes the two blocks of content that, on a PC browser, would appear side by side and places them one on top of the other. Group 1 displays above group 2. The tag `<mm:id-ref>` uses its `idref` attribute to reference IDs of the pre-defined groups.

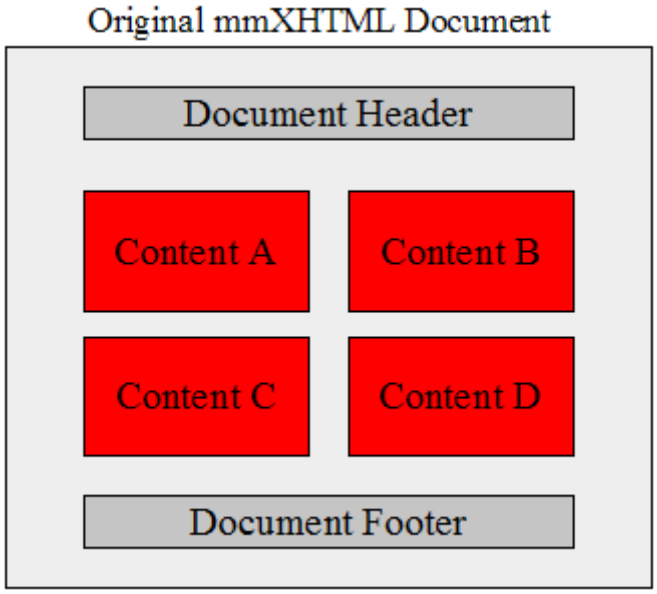
Method 2: Use structures for PDA display

In the last example, the <mm:layout> tag was used to specify a file that contained groups of content that have been reorganized into a simpler layout. Suppose, however, instead of two groups in *myFile.jsp*, there were ten or more. Reorganizing the content using the preceding method would result in a very long page that would require scrolling to see the entire contents. It might also result in the page being split in an inappropriate place if the attempt at displaying the single page exceeds the memory limitations of the device.

PDA pagination allows the content authors to split a single page into multiple pages so that it can be viewed more easily on a PDA device. Along with the sub-pages, persistent navigational components are automatically created to help the user navigate around the pieces of the document.

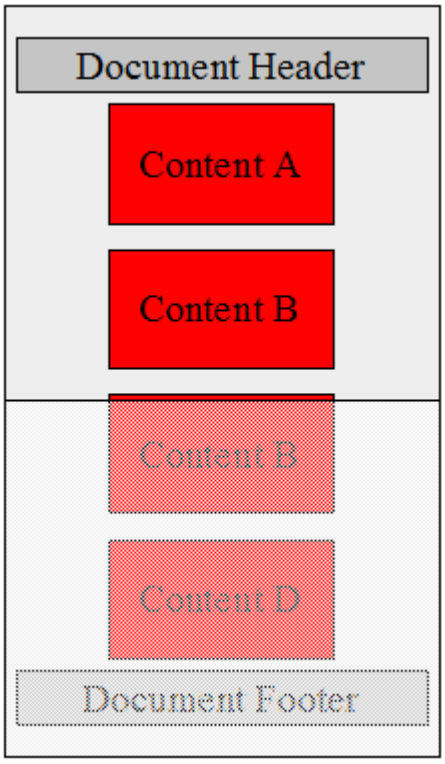
The following example shows a typical layout design for content targeting a PC browser. A document header and footer surround four logical content blocks – perhaps news stories, or product features.

Typical Web Layout for Display on a PC Browser



The next illustration shows how this content might be organized using a layout file containing a simplified template, as was seen in the previous example using *myFile.jsp*. For larger pages, parts of the content will not be visible without scrolling.

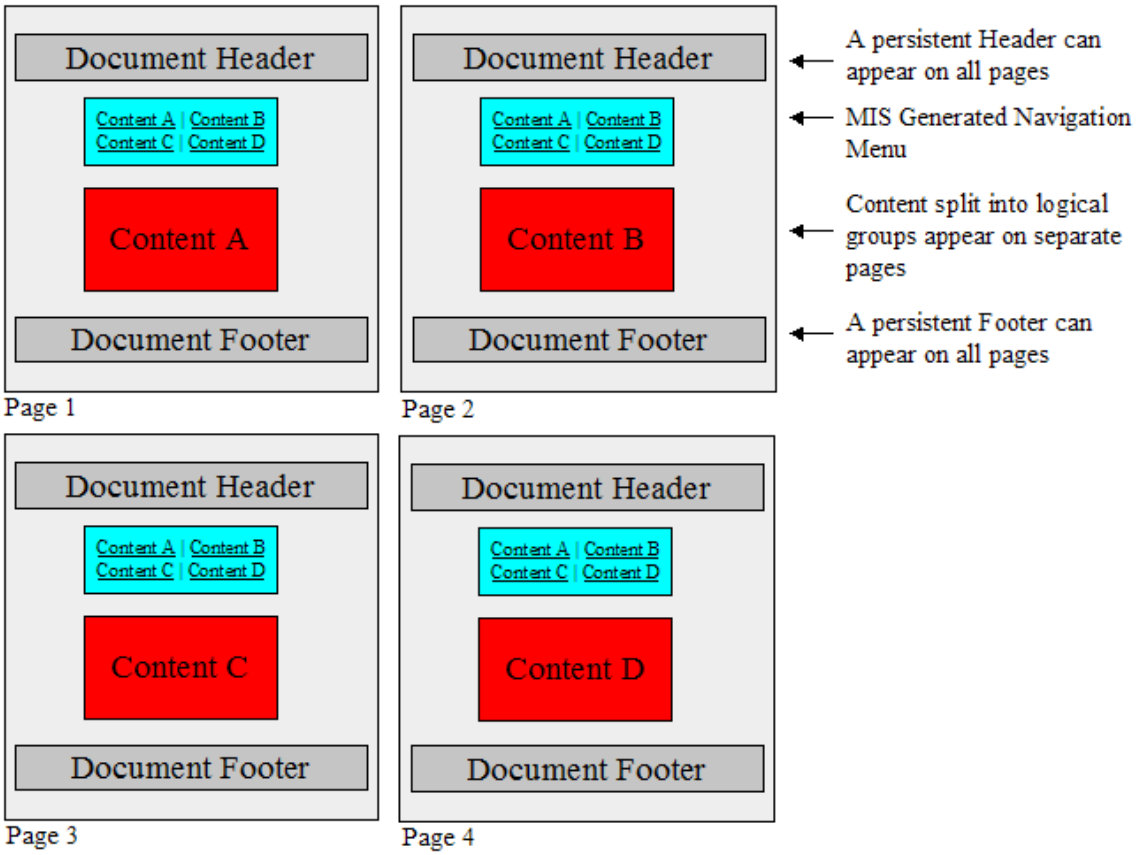
Ogranizing Content Using a Layout File



For smaller amounts of content, this solution works well. For larger amounts of content, pagination works better.

The following graphic illustrates how the four content blocks might appear if the content was paginated using the `<mm:structure>` tag.

Splitting Content into Pages with Navigation on a PDA



The following code blocks illustrate how to paginate content intended for PDA display using the `<mm:structure>` tag. This example uses a layout file in which to place the `<mm:structure>`. This is generally considered good practice as it keeps separate the content file and the presentation file.

pagination.jsp

```
<%@ taglib uri="mmJSPtaglib" prefix="mm"%>
<mm:page content="false">
<html>
<head>
<title>Pagination</title>
<link rel="stylesheet" href="mystyle.css" type="text/css"/>
<h4><strong>PDA Pagination Demo</strong></h4>
<mm:layout src="pagination_layout.jsp" where="IsPDA"/>
</head>
<body>
 <table>
    <tr><td>
  <mm:group id="groupA" title="Hamlet">
     <p>There are more things in heaven and earth, Horatio, than are dreamt of
in your philosophy.
</p>
  </mm:group>
  </td></tr>
  <tr><td>
```

```
  <mm:group id="groupB" title="As You Like It">
     <p>All the world's a stage,
And all the men and women merely players;
They have their exits and their entrances,
And one man in his time plays many parts,
His acts being seven ages.
</p>
  </mm:group>
  </td>
  </tr>
  <tr><td>
  <mm:group id="groupC" title="Macbeth">
     <p>Life is but a walking shadow,
a poor player that struts and frets
his hour upon the stage and then is heard no more.
It is a tale told by an idiot,
full of sound and fury signifying nothing.
</p>
  </mm:group>
  </td></tr>
  <tr><td>
  <mm:group id="groupD" title="Twelfth Night">
     <p>Be not afraid of greatness. Some are born great, some achieve
greatness, and some have greatness thrust upon 'em.
</p>
  </mm:group>
  </td></tr>
</table>
</body></html></mm:page>
```

pagination_layout.jsp

```
<%@ taglib uri="mmJSPtaglib" prefix="mm"%>
<mm:page content="false">
<html>
<head>
<title>Pagination</title>
</head>
<body>
  <mm:structure id="pagination_str" where="IsPDA">
     <mm:group-ref idref="groupA" depth="0" display="headings" type="normal"/>
     <mm:group-ref idref="groupB" depth="0" display="headings" type="normal"/>
     <mm:group-ref idref="groupC" depth="0" display="headings" type="normal"/>
     <mm:group-ref idref="groupD" depth="0" display="headings" type="normal"/>
  </mm:structure>
  <br/>
  <h5>Copyright &copy; BEA Systems, Inc.</h5>
</body>
</html>
</mm:page>
```

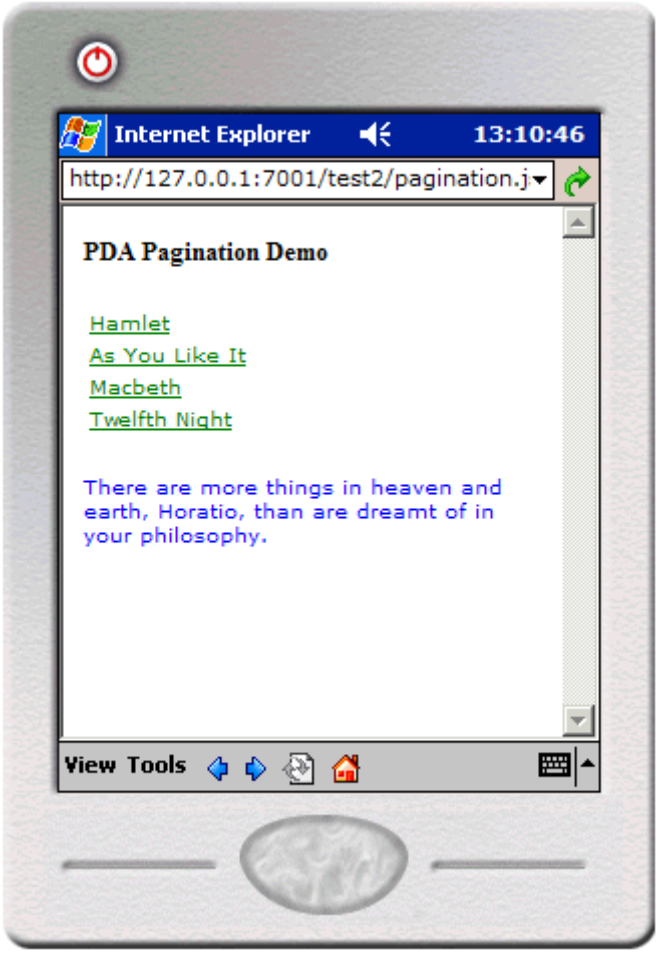
A style sheet has also been added to this example to distinguish the different parts of the layout.

mystyle.css

```
p {
       font-family: Verdana, Arial, Helvetica, sans-serif;
       font-size: 10pt;
       color: blue;
}
h4 { color: black; font-size: 10pt; }
h5 { color: gray; font-size: 8pt; }
a:link {color: green; }
```

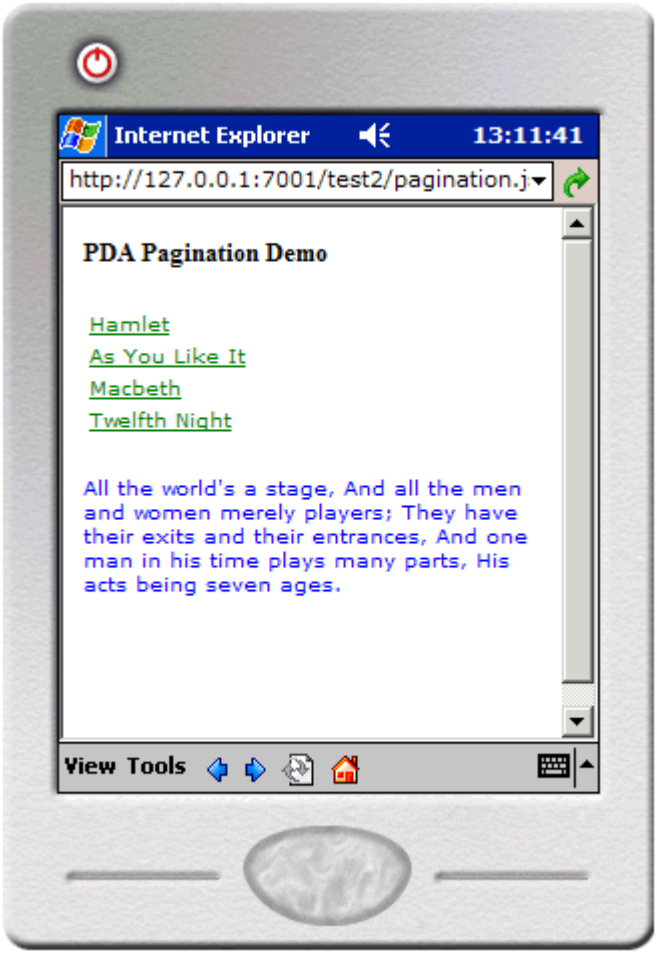
As can be seen on the next page, a navigation section replaces the `<mm:structure>` and only one group is displayed at a time.

Displayed Navigation Section



Clicking on a link shows the contents of that group. Content outside the `<mm:structure>` is persistent, meaning it will appear on all pages.

Display Group Contents



The menu links are created from the text contained in the group's head. If the group has no head, the group title is used instead.

The groups used to form the navigation structure are referenced from within the `<mm:structure>` tag using the `<mm:group-ref>` tag.

Persistent content, such as the header and footer in the preceding example, can be created by placing `<mm:id-ref>` tags in the layout page, but outside the `<mm:structure>`.

The `<mm:group-ref>` attributes used for PDA pagination are described as follows:

- `idref`**:** The `idref` attribute refers to the ID of the group to be referenced
- `depth`**:** Because each group in this example has no sub-groups, the `depth` attribute is set to 0. Each group referenced by an `<mm:group-ref>` is searched at the depth specified in the `<mm-group-ref>` using the `depth` attribute. The navigation section displays a link to each group down to the level that has been specified
- `type`="normal" (This is the default if this attribute is missing.) For PDA pagination, "normal" is the only acceptable value for this attribute
- `display`="headings" For PDA pagination, this is the only acceptable value for this attribute

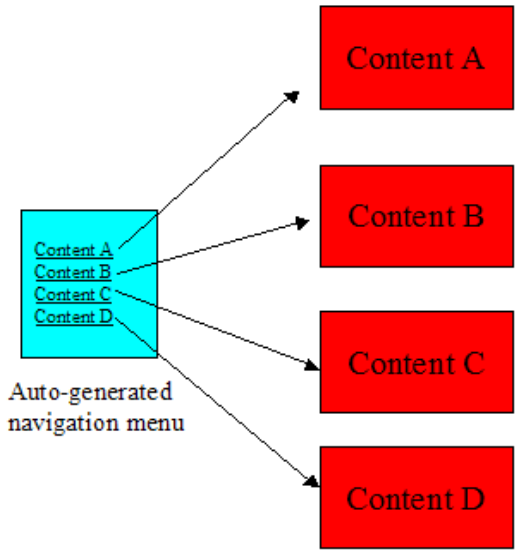
Notes:

- For PDA pagination, if either of these last two attributes is not set as per above, a fatal exception will occur
- Multiple structures can be placed in a single file, however only the first structure found containing a `where` attribute that matches the requesting device is used
- The structure can be placed either in the main file or a layout file but if you are using a layout file for PDAs, the structure used for PDA pagination must be placed within that layout file

Restyle for Menu-Driven Devices

The screen size and memory capabilities are even more restricted on menu-driven devices such as phones. As a result, it is usually always necessary to split web content into smaller units in order to present it in an easy-to-read fashion. This restructuring is done with the `<mm:structure>` and `<mm:group-ref>` tags, using the same technique that was introduced in the preceding section on PDA pagination. The`<mm:group-ref>` tag contains attributes that allow a high level of control over how content is presented on menu-driven devices.

WML Pagination Using <mm:structure>



Unlike content delivered to PDAs using the `<mm:structure>` tag, only content referred to within the `<mm:structure>` itself is delivered to menu-driven devices. Content outside the structure will be ignored. The preceding PDA pagination example sent a page header and footer to the device although they were outside the `<mm:structure>`. If the same layout file were sent to a menu-driven device, the header and footer would not appear.

Using the preceding example and changing the *where* attribute of the `<mm:layout>` and `<mm:structure>` tags to target menu-driven devices:

```
…where="IsMenuDriven"
```

results in the navigation appearing on a page separate from the content on a menu-driven device.

Navigation Appears on Page Separate From Content

Image Courtesy of Openwave Systems Inc

Clicking on a link shows the content of the group on its own sub-page.

Displaying Content of the Group

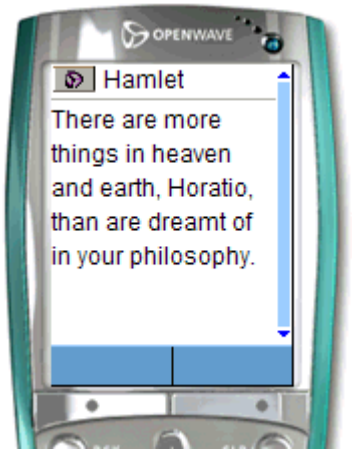


Image Courtesy of Openwave Systems Inc

Notice that the header and footer material does not appear. Only the groups referenced from within the structure appear on a menu-driven device.

Use <mm:group-ref> Attributes for Menu-Driven Display

The format of the `<mm:group-ref>` tag is:

```
<mm:group-ref idref="..." type="..." depth="..." startdepth="…" display="…"
navstyle="…"/>
```

Note: This tag does not have a separate closing tag, so a slash must be placed before the final angle bracket to make it “well-formed”.

The `<mm:group-ref>` attributes used for menu-driven display are described as follows:

- `idref`: The attribute `idref` is given the same value as the `id` of the group that is to be referenced.
- `type`: The attribute `type` can have the values "normal" or "options". When content is being delivered to a WML device, this attribute specifies whether the menu displays on the screen or whether it displays behind the **Options** button on the device. Setting the value to “options” has no effect on XHTML MP devices. (Remember that “options” is not used for PDA pagination either.)
- `depth`: The attribute `depth` controls the number of levels you want in the hierarchy. This applies when you have groups nested within groups (that is, subgroups). The possible values are "flat", 0, 1, or 2:
  - flat: displays the entire contents of the referenced group
  - 0: displays a link to the parent group
  - 1: displays a collection of links to the parent groups and the immediate child groups
  - 2: displays a collection of links to the parent and all nested sub-groups
- `startdepth`: The attribute `startdepth` can have an integer value. It specifies the depth at which the navigation list starts. For example, if you have a group hierarchy 3 deep, that is, a group with a sub-group which in turn has another subgroup, specifying startdepth="1" and depth="1" will return just the first level of subgroups. This is used primarily when you do not want an outer group to appear as a link in the hierarchy
- `display`: The attribute `display` can have the values "all", "headings" or "links"
  - all: displays everything within the group - headings, bodies, and so on.
  - headings: displays the headings within a group down to the level (depth) specified
  - links: displays headings and any links that occur within the group

Note: To display the entire contents of a group, set `depth="flat" display="all".` By putting the header of the preceding example into the structure and setting these attributes, the text of the group will appear in its entirety, rather than as a link.

```
<mm:structure id="pagination_str" where="IsMenuDriven">
   <mm:group-ref idref="header" depth="flat" display="all" type="normal"/>
   <mm:group-ref idref="groupA" depth="0" display="headings" type="normal"/>
   <mm:group-ref idref="groupB" depth="0" display="headings" type="normal"/>
   <mm:group-ref idref="groupC" depth="0" display="headings" type="normal"/>
   <mm:group-ref idref="groupD" depth="0" display="headings" type="normal"/>
</mm:structure>
```

Displaying the Content of the Group

Image Courtesy of Openwave Systems Inc

For a complete list of possible combinations using `depth` and `display`, see "Appendex A – Mobility Tag Reference."

Note: The attribute `navstyle` is used to style the navigation block in this example. This attribute is covered in detail in the next section.

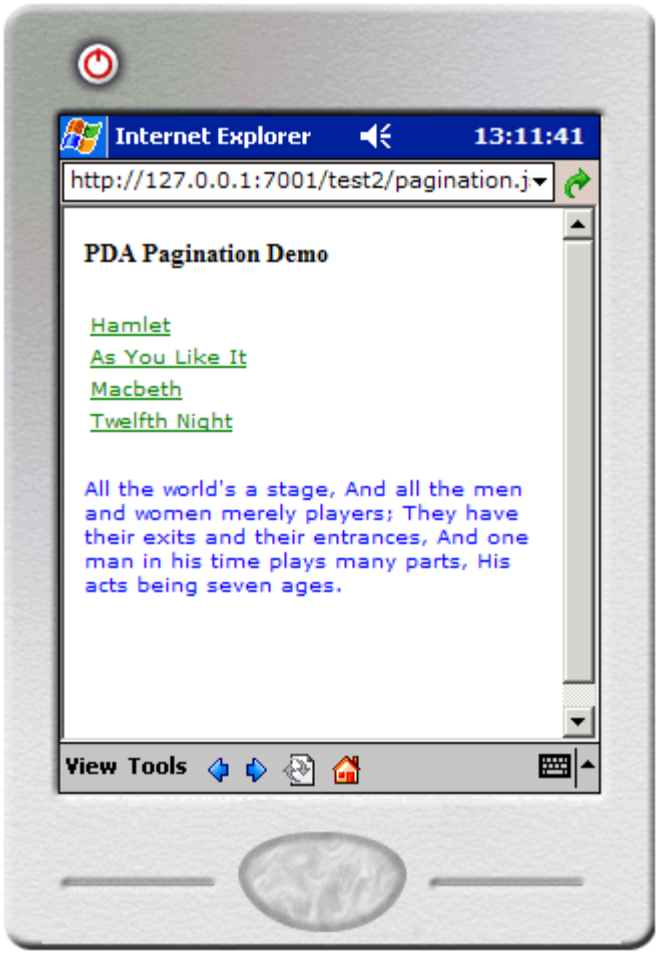
Navigational Menu Styling

The previous section showed how WebLogic Mobility Server creates navigational structures for PDA and menu-driven devices. This section shows how to style these navigational structures.

Without any styling, the navigation created by WebLogic Mobility Server displays as a single column table as can be seen in the following illustration.

Navigation menus created by WebLogic Mobility Server will default to a single column table style if no other style directions are given.

Default Single Column Table Navigation Styling



WebLogic Mobility Server allows content authors to customize the navigation.

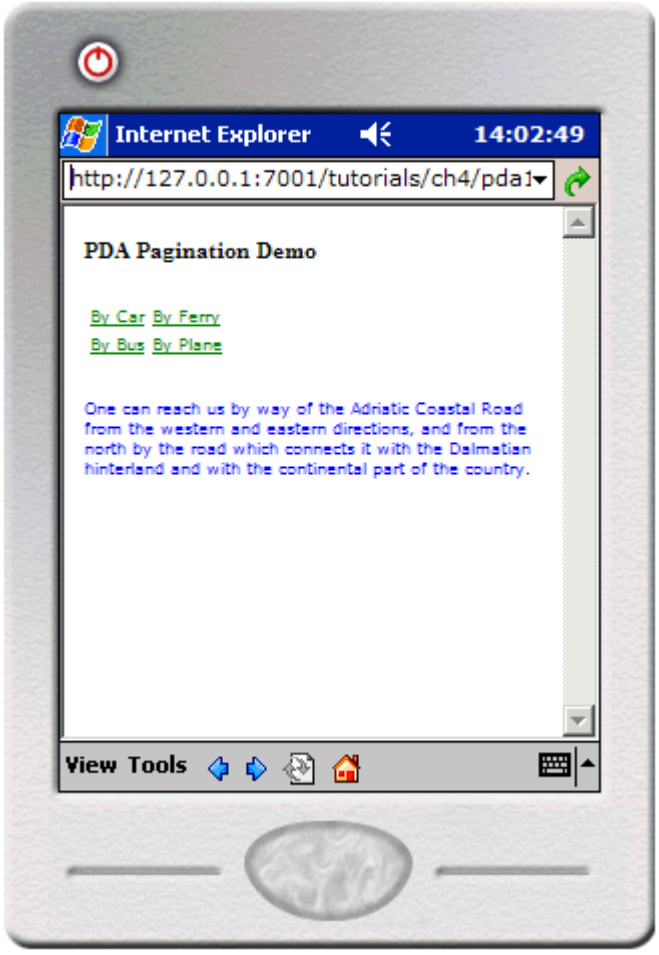
The following types of styling are possible:

- Multi-column table with the user able to set the number of columns and rows
- List with items separated by
- In-line styling showing either space-delimited or pipe-separated lists
- Images used as navigational links, with or without accompanying text

Style Navigational Menus Using Multi-Column Tables

The following illustration shows an example of a navigation menu restyled using a two-column table.

Two-column Styling of Navigation Table



To get the two-column navigational table to display on PDAs and menu-driven devices that support tables, set the `<mm:structure>` attribute `navstyle` as follows.

```
<mm:structure id="s1" where="IsPDA" navstyle="nav-format:table;
nav-table-columns: 2">
```

You can also set the number of rows by using `nav-table-rows: n` in the `navstyle` attribute where `n` is the number of rows.

```
<mm:structure id="s1" where="IsPDA" navstyle="nav-format:table;
nav-table-rows: 2">
```

Note: If there aren't enough links to fill the columns or rows that have been indicated in the `nav-table-rows` property, empty rows or columns will be added.

If the request is made from a device that does not support tables, the navigation menu is rendered as a list. All list elements are displayed on the same level, even if sub-groups are involved.

Table styling on a device that does not support tables is ignored. The menu items are rendered as a list.

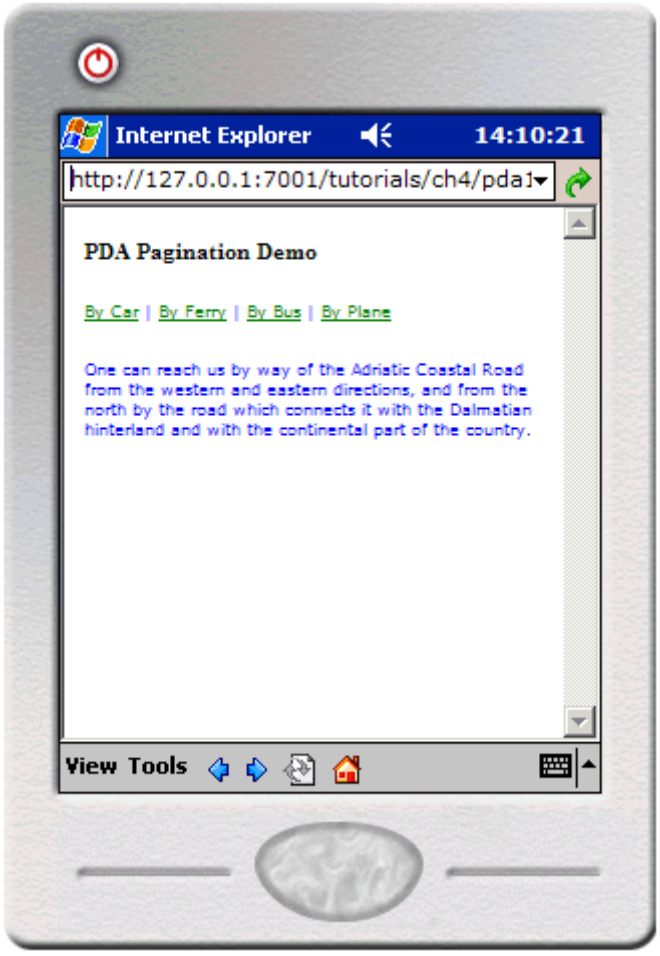
Table Styling for Device Without Table Support

Style Navigational Menus Using Lists

An alternative to displaying menu items in a table is to display them in a list. For this, the `navstyle` attribute value `nav-format:list` is used. This can be used to create a list of menu items separated by a line-break, a space, or a pipe character.

Using lists can be helpful when delivering content to devices with vastly different screen widths. WebLogic Mobility Server will retrieve the width of the requesting device from its Device Repository and wrap the list in an appropriate place. Using tables can have undesirable consequences if the screen width can't accommodate the width of the table. Excessive wrapping of text within cells can occur making the links less readable.

Navigation Styling



To use a list, add the following values using the `navstyle` attribute of the `<mm:structure>` tag.

- For a <br/> separated list:
  navstyle="nav-format: list"
- For a space-separated list:
  navstyle="nav-format: list; nav-list-item-display: inline"
- For a pipe-delimited list:
  navstyle="nav-format: list; nav-list-style-type: pipe"

Style Navigational Menus Using Images

Images can be used to style menus delivered to PDAs and menu-driven devices. Icons can be inserted as bullets or be made into links themselves.

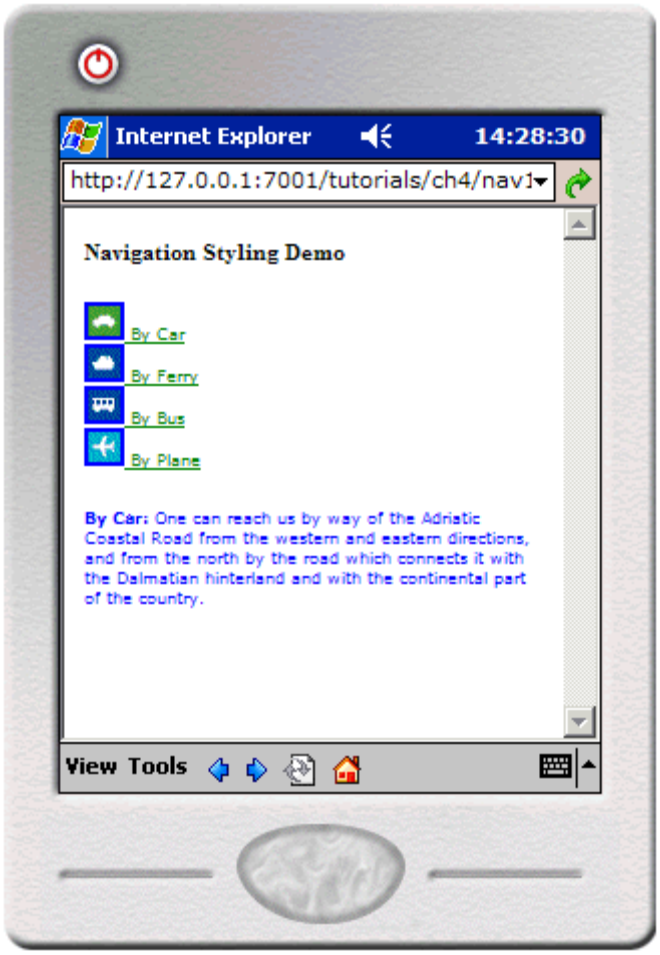
Using `<mm:media-group>` to display images

Images that are being delivered to different device types should be placed inside an `<mm:media-group>` element in order to deliver the best image to each device. The media-group is also used in navigational styling to suppress the immediate delivery of the image in the place where the `<mm:img>` mark-up is located in the document. The images are used in the menu creation only.

To do this, set `navstyle="display:none"` on the `<mm:media-group>`. The attribute `id` is used so that the images can be referenced from other places in the mark-up.

```
<mm:media-group id="car" navstyle="display:none" alt="*">
   <mm:img where="ImgGIFSupported" src="img/car.gif" alt="*"/>
   <mm:img where="ImgWBMPSupported" src="img/car.wbmp" alt="*"/>
</mm:media-group>
```

Navigation Styling with Images



Adding images to navigation is done using the `navstyle` attribute in any of the following tags:

- <mm:group>
- <mm:group-ref>
- <mm:structure>

The three tags have a hierarchical relationship. Contradicting styles will be decided depending on where the tags appear in the hierarchy. So, for example, a navstyle attribute on an <mm:group> will override that of an <mm:group-ref> or <mm:structure>. The navstyle of an <mm:group-ref> will override the navstyle of the <mm:structure>.

The navstyle attribute can also be inherited from a parent tag. This means, for example, that if the author specifies a URL on an <mm:structure>, this will be applied to all groups that are referenced from within the structure unless otherwise overridden.

The navstyle attribute is used to refer to the media-groups that contain the images to be used in the navigation menus. In the following example, an image of a car has been placed inside a media-group. The id attribute of this media-group has set to “car”. The group-ref then references this image from within its navstyle attribute.

```
<mm:group-ref idref="groupA" depth="0" display="headings" type="normal"
navstyle="nav-image:url(#car)"/>
```

Style Navigation Text and Images

Styling the menus with images and text is done with the nav-text-display style. This can have three values:

- inline
- none
- block

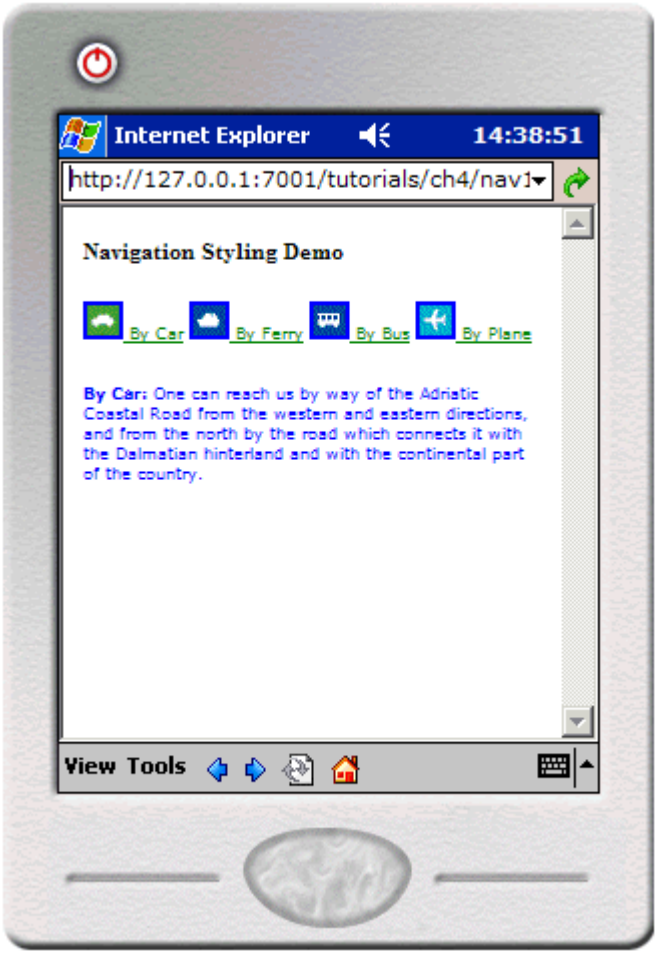
Examples

The following examples illustrate the use of these values.

Example 1: Images with Inline Text Styling. The text displays next to the image (this is the default).

```
<mm:structure id="str1" where="IsPDA" navstyle="nav-format:list; nav-list-item-display: inline; nav-text-display: inline">
```

Navigation Styling with Images and Inline Text



Example 2: Images with No Text Styling. Images appear as links by themselves.

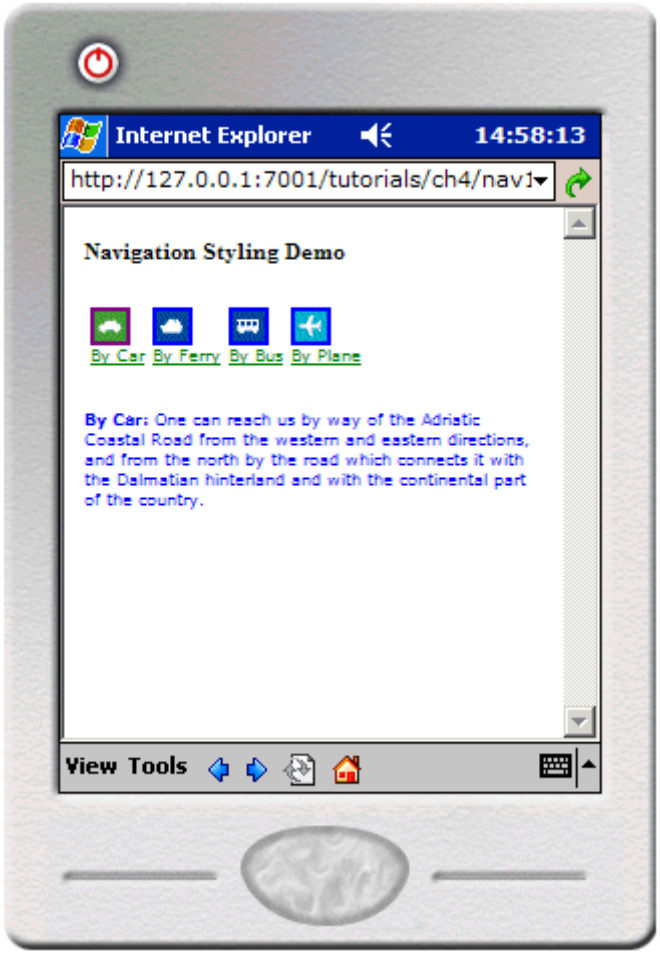
```
<mm:structure id="str1" where="IsPDA" navstyle="nav-format:list; nav-list-item-display: inline; nav-text-display: none">
```

Navigation Styling with Images and No Text

Example 3: Images with Block Text Styling. Text displays beneath the images in a table.

```
<mm:structure id="str1" where="IsPDA" navstyle=" nav-format:table; nav-table-
rows: 1; nav-text-display: block">
```

Navigation Styling with Images and No Text



Menu Design Tips

The first of the three preceding illustrations uses a two-column table to avoid awkward wrapping that would result in three menu items on the first row and one on the second. This highlights an issue that content developers should keep in mind. Deciding on whether to use tables or lists to style navigation has to be thoroughly considered. Each has its advantage.

Tables can be styled with borders and backgrounds (if the device supports these things). The drawback, however, is accommodating the varying screen widths of the smaller devices. What may look good on a Nokia 3650 may wrap excessively on a Sharp GX10. On some browsers, exceeding the width of the screen will result in the table being automatically reduced to a single column.

Lists have a bit more flexibility. If the browser supports wrapping, the menu items will wrap automatically when the maximum screen width of the requesting device is reached.

Using images as menu links with no additional text below or beside the image works well in a list. The images, however, must be equal in size.

The `<mm-structure>` should have the following `navstyle` attribute properties:

```
navstyle="nav-format: list; nav-list-item-display: inline; nav-text-
display:none"
```

The result is that the browser will “wrap” the list of icons to fit the screen. The benefit of this is that the content author doesn't have to “hard code” the number of columns or rows; they can rely on the browser to decide.

This method generally gives the most visually appealing result across a range of devices.

Note: Try to keep text strings used in navigation to as short a length as possible.

More Navigation Lists for Handheld Devices

Up to this point, the navigation lists described have been created using pre-defined groups. Selected groups on a single page have been referred to from within an `<mm:structure>`. The group headings form the link text and clicking on the link takes the user to the content of that group. Sometimes, however, you might want to create a navigation list whose items are not based on any pre-defined group. WebLogic Mobility Server provides two tags to let you do this.

- `<mm:nl>`
- `<mm:li>`

These two tags are roughly the equivalent of the XHTML 2.0 tags `<nl>` and `<li>`. Styling attributes can be added to each of these tags in order to manipulate the navigation to best suit the requesting device.

The following is an example of how these tags can be used.

```
<div style="border: 1px solid">
<mm:nl navstyle="nav-format: table; nav-table-columns: 2" where="IsPDA">
   <mm:li navstyle="nav-image: url(clubs.gif)" href="clubs.htm">Clubs</mm:li>
   <mm:li navstyle="nav-image: url(diam.gif)" href="diam.htm">Diamonds</mm:li>
   <mm:li navstyle="nav-image: url(spades.gif)"
    href="spades.htm">Spades</mm:li>
   <mm:li navstyle="nav-image: url(hearts.gif)"
    href="hearts.htm">Hearts</mm:li>
```

```
</mm:nl>
</div>
```

The `navstyle` attribute is used in the same way as the preceding examples that use structures. The difference is that the `<mm:li>` element refers to a URL using the `href` attribute rather than a specific group referenced by the `<mm:group-ref>` element. In this example, the URL points to a particular file (for example `navstyle="nav-image: url(clubs.gif)"`). As with the preceding examples, a media-group can be referenced instead (for example `navstyle="navimage: url(#clubs)"` where “clubs” is the `id` of a media-group. For even more flexibility, the `where` attribute can be used with the `<mm:nl>` element. This means that developers can refine the menu for specific targeted devices. As is the case with the where attribute of `<mm:structure>`, where="IsFullBrowse " is not allowed. Additional styling can be added using external style sheets as described in “Work with Style Sheets.”

Work with Style Sheets

This section describes:

- What style sheets are and how they are used
- The advantage of server side cascading style sheets (SSCSS) for delivering pages to PDAs
- Examples of HTML style sheets
- How to use SSCSS
- Using the default style sheet
- An example of SSCSS with a sample file

Introduction to Style Sheets

A style sheet is a simple mechanism for adding style (for example fonts, colors, spacing) to Web documents.

The World Wide Web Consortium (W3C) has actively promoted the use of style sheets on the Web since the Consortium was founded in 1994. The W3C has made several recommendations including CSS1, CSS2, XPath, XSLT and XSL. CSS (Cascading Style Sheets) especially are widely implemented in browsers.

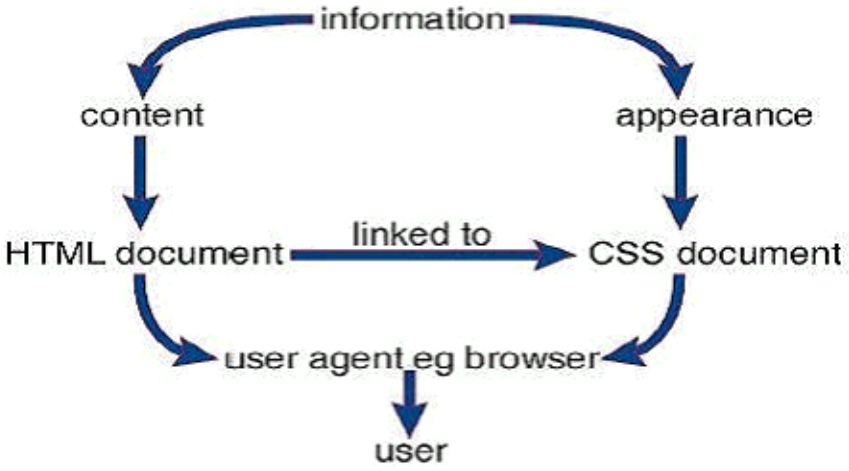
By attaching style sheets to structured documents on the Web (for example HTML), content authors and web developers can influence the presentation of documents without sacrificing device-independence or adding new HTML tags.

Style sheets can be used to define the appearance of an entire site in a consistent way. With the introduction of CSS, it is now recommended that layout-specific features in HTML be phased out and replaced by style sheets.

Understand Cascading Style Sheets

Cascading Style Sheets (CSS) provide a means for web authors to separate the appearance of web pages from the content of web pages. This means that the content of the web site should go into your HTML files (or XHTML files or JSP pages), but these files should not describe how that information is displayed. Information about how the pages should appear goes into CSS files. The styles from this file that is given a .css extension are then applied server-side.

Keeping Document Content and Style Separate



In addition to being in *external.css* files, style sheets can also be placed internally in the `<head>` element of the content file.

A simple example is shown here:

```
body {
{font-family: arial, Helvetica, sans-serif;
font-size: 1em;
text-align: justify}

h1
{font-family: arial, sans-serif;
font-size: em}

p
{font-family: Courier New, sans-serif;
font-size: 1em}

.note
{background-color: #003333;
}
```

Rules In CSS

A rule is a statement that tells a browser how to draw a particular element on a web page.

A rule has two parts: a selector and a declaration. A selector identifies the elements on a web page that are affected by the rule. The declaration tells the browser how to display the element that is selected by the selector.

The preceding example has four rules. There are four selectors: body, h1, p and note. The declarations for each rule are inside curly braces. Each declaration can contain one or more properties. A semi-colon separates properties.

Link Style Sheets

External style sheets can be applied to multiple documents. Each document must be linked to the style sheet in order for the styles to be applied. Placing a link to the style sheet in the head of the marked up content file does this. When the browser begins reading the page, it sees the style sheet link, and downloads the style sheet, then uses it to display the page.

To link a web page to a style sheet, place a link to the *.css* file in the head of the document, using the following syntax:

```
<link rel="stylesheet" type="text/css" href="mystyles.css"/>
```

The style sheet should be accessible to all files that use it. Typically, it is placed in the root directory of the web folder.

Server Side Application of CSS for PDAs

If a style sheet is unavailable, the requesting browser is responsible for applying any internal or inline styles that are contained in the content file. In the absence of any styles, the browser will use its own default settings. Some PDA browsers, however, cannot support CSS. In cases like this, WebLogic Mobility Server will attempt to apply any external or internal styles by translating them into the nearest equivalent in-line style. The server does this translation before the page is delivered to the browser.

Use Multiple Device-Specific Style Sheets

When creating a style sheet you can create a single style sheet that applies to all devices, or you can create multiple specialized style sheets that target particular device types.

Creating multiple style sheets can be easier to maintain and can allow authors to finely tune content presentation to particular devices. You might decide, for example, to present web content that is being requested by handheld devices in a smaller font than when it is being requested by a desktop browser. Specifically, headings can be displayed in a very large font on a desktop browser, but the smaller screen devices would likely cause this heading to wrap awkwardly.

Multiple style sheets linked to in the document head using the `<mm:include>` mobility tag.

```
<mm:include where="UsableWidthPixels > 180">
  <link rel="stylesheet" type="text/css" href="mytinystyles.css"/>
</mm:include>
<mm:include where="IsLandscapePDA">
  <link rel="stylesheet" type="text/css" href="mywidestyles.css"/>
</mm:include>
```

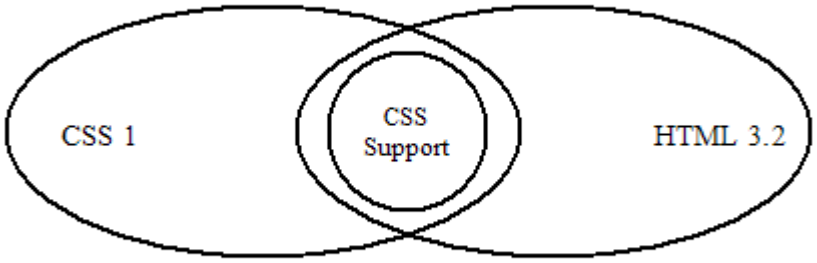
WebLogic Mobility Server CSS Support

Not all CSS styles can be supported across all devices. This means that WebLogic Mobility Server will support the elements that are common between CSS1 and the mark-up language of the requesting device.

Many PDAs support HTML 3.2. The following graphic represents how the common elements within the CCS1 specification and the HTML 3.2 specification are included as part of WebLogic Mobility Server SSCSS support.

WebLogic Mobility Server supports the common elements between CSS1 and the requesting device.

WebLogic Mobility Server CSS Support



Styles Available in HTML 3.2

The following tables define the CSS properties supported by the WebLogic Mobility Server targeting HTML 3.2 browsers.

CSS1 to HTML 3.2 Property Relationship

Property	Supported Values	Elements
font-family	<family-name> <generic-family>	Any
font-style	normal | italic	Any
font-weight	normal | bold	Any
font-variant	normal | small-caps	Any
font-size	<absolute-size> | <relative-size>	Any
background-color	<color>	<body>
background-image	URL (for example url("../images/image.jpg")	<body>

color	<color>	<body>, <font>
font	<font-family> <font-style> <font-weight> <font-size> <color>	Any
text-decoration	underline linethrough	Any
vertical-align	[top | middle | bottom] | [top | bottom]	<caption>, <tr>, >th>, <td>, <input>, <img>
text-align	left | right | center	<h1>, <h2>… >h6> <p>, ,
a.link (pseudo selector)	<color>	<body>
a.visited (pseudo selector)	<color>	<body>
a.active (pseudo selector)	<color>	<body>
list-style	inside	, , <dl>, <dt>, <dir>, <menu>
list-style-type	[disc | circle | square] | [decimal | lower-roman | upper-roman | lower-alpha | upper-alpha]	,
height	<length>	<hr>, <th>, <td>, <img>
width	<length> | <percentage>	<hr>, <table>, <th>, <td>, <img>
border	<border-width>	<table>, <img>
padding	<length>	<table>
margin	<length> | <percentage>	<img>

HTML 3.2 to CSS/SSCSS Property Relationship

Element	Element Attribute	CSS/SSCSS Property
<body>	bgcolor	background-color
	background	background-image
	link	a.link (pseudo selector)
	vlink	a.visited (pseudo selector)
	alink	a.active (pseudo selector)
<h1>, <h2> and so on (headings)	align	text-align
<p>, ,	align	text-align
	type	list-style-type [disc | circle | square]
	type	list-style-type [decimal | lower-roman | upper-roman | lower-alpha | upper-alpha]
<dl>, <dt>, <dir>, <menu>		
<pre>		
<hr>	align	text-align
	size	height
	width	width
<table>	align	horizontal-align
	width	width
	border	border
<caption>	align	vertical-align
<tr>	align	horizontal-align
	valign	vertical-align
<th>, <td>	align	horizontal-align
	valign	vertical-align

	width	width
	height	height
<input>	size	chars
	align	horizontal-align OR vertical-align
<img>	align	horizontal-align OR vertical-align
	width	width
	height	height
	border	border
<font>	size	font-size
	color	color
	face	font-family

Define Colors in HTML 3.2

To specify colors, you can use the following keyword color names:

Keyword Color Names

Color	**Color**	**Color**	**Color**
aqua	gray	navy	silver
black	green	olive	teal
blue	lime	purple	white
fucshia	maroon	red	yellow

These 16 colors are taken from the Windows VGA palette. You can use any color specification you want, for example, hex, RGB, such as:

```
p{ color: rgb(255,255,255); } or p{ color: rgb(fff);} or p{ color: #ff0000;}
```

Manage Navigation

This section describes pagination and Back-to-Top functionality in WebLogic Mobility Server.

Pagination

Pagination refers to dividing a document into pages. WebLogic Mobility Server provides the following two pagination options:

- **Automatic pagination**

  Authors need not worry about manipulating pages, cards or decks (groups of cards) because WebLogic Mobility Server automatically handles pagination for targeted devices.

- **Author-controlled pagination**

  WebLogic Mobility Server provides the option of author-controlled pagination. Author-controlled pagination overrides the automatic pagination function within WebLogic Mobility Server.

Automatic Pagination

WebLogic Mobility Server automatically divides mmXHTML or the WebLogic Mobility Server JSP taglib documents into pages of a size appropriate to the device receiving the content.

Automatic Pagination Rules

In WebLogic Mobility Server, content elements are given default pagination rules. The following pagination rules apply:

- Page breaks are avoided inside words so that a word does not begin on one page and finish on another
- Page breaks are avoided inside elements, such as tables, forms and paragraphs
- Page breaks are positioned so that small chunks of content do not occur on a new page or card. For example, a page does not start with:
  `<p>a few words. The End.</p>`
- If an HTML element or <mm-group> element is too big to fit on a page then that element is split, for example, at the end of a sentence

Author-Controlled Pagination

This section describes how the author can override automatic pagination by specifying pagination controls in the document.

Author controlled pagination is the splitting of content by the author onto separate pages or cards. The main objectives of the author-controlled pagination are as follows:

- To provide authors with the ability to control where page breaks occur
- To provide authors with the ability to control which elements should not be separated

This functionality uses concepts and syntax defined in the CSS 2 paper on “Paged Media” as defined by the World Wide Web Consortium (W3C):

http://www.w3.org/TR/REC-CSS2/page.html

Authors can set pagination style on all HTML elements and the mmXHTML element <mm-group> in order to alter the default behavior of automatic pagination. The <mm-group> element is used to chunk mmXHTML content into groups. WebLogic Mobility Server does not split groups into separate pages

There is no need to set pagination style on the other mmXHTML tags as these do not behave as HTML elements.

Use of Page Break Avoid

The use of the page-break-avoid controls overrides situations in which an automatic page break would occur.

Example:

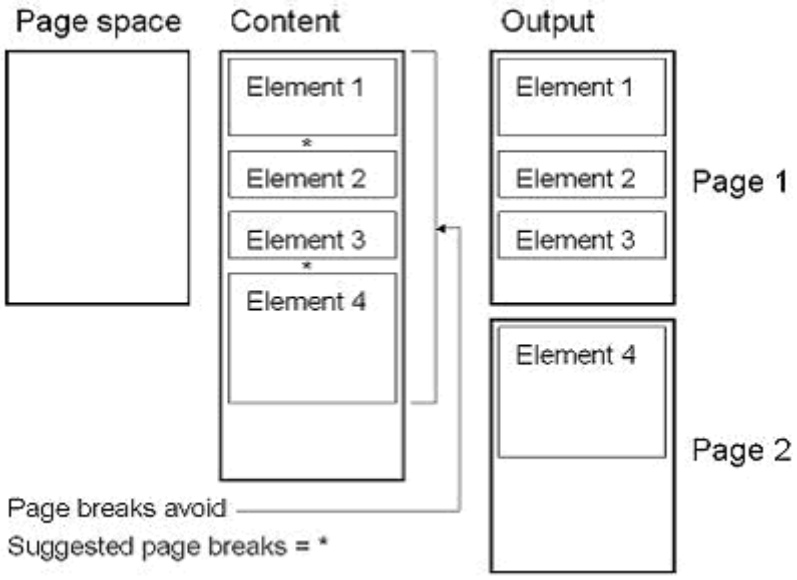
The author indicates that an element should be kept together:

```
<div style="page-break-inside: avoid">
```

This is a long paragraph

```
</div>
```

Placing Page Breaks in a Document



Use of Suggested Page Break

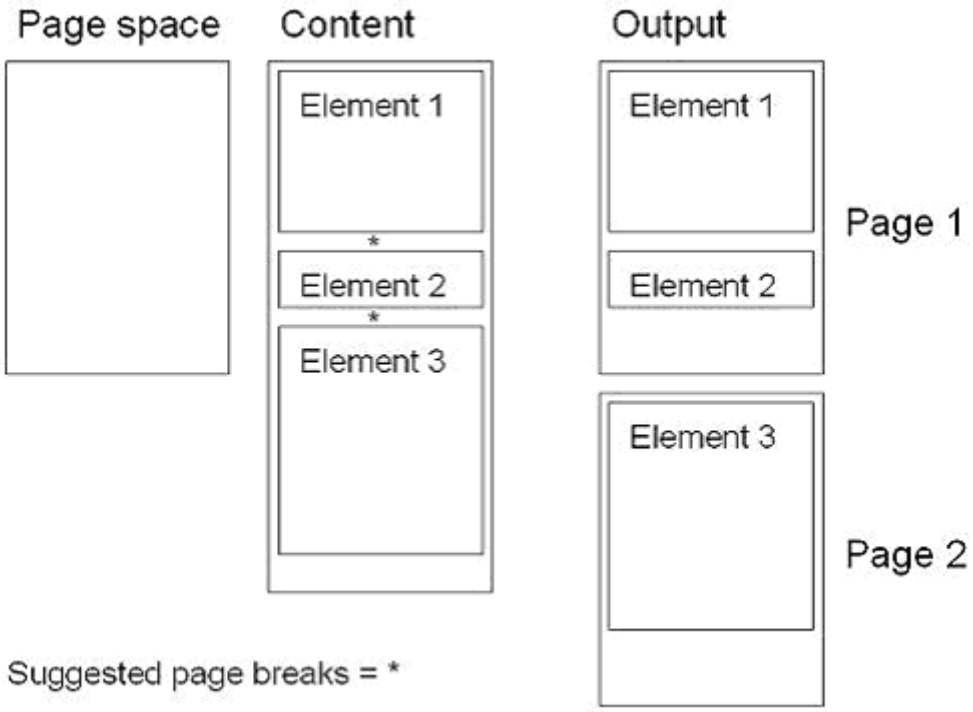
The author can “suggest” that a page break should occur before or after a HTML element or mmXHTML or the WebLogic Mobility Server JSP taglib group. If an automatic page break is going to occur in the vicinity of a suggested page break the suggested page break will be used instead. Suggested page breaks make navigation easier.

Example

```
<p style="page-break-after: suggested">Paragraph One</p>
<p style="page-break-after: suggested">Paragraph Two</p>
<p>Paragraph Three</p>
```

In this example, two page breaks have been suggested. One before paragraph two and one after. The result will be that the page will break after paragraph two. The reason is that WebLogic Mobility Server, after reaching the maximum page size, will search backwards through the content and break at the nearest suggested page break.

Suggested Page Breaks



Controls Applied to Multiple Elements

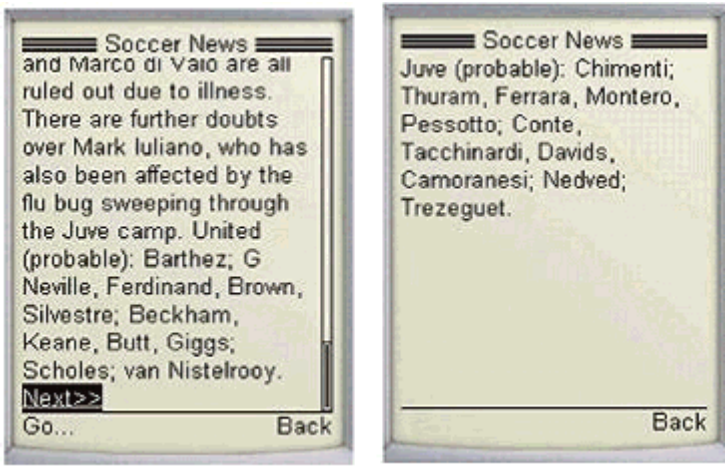
WebLogic Mobility Server attempts to implement pagination controls that apply to multiple elements before implementing controls on individual elements. The following example illustrates this. A page break avoid has been applied to elements 1, 2, 3 and 4. WebLogic Mobility Server attempts to keep these elements together. However, the four elements do not fit into the page space of a particular device. In this case, WebLogic Mobility Server checks for controls applied to individual elements and implements the suggested page break that is closest to the end of the page.

Page Break Example

In the following example, a page break is suggested after the first paragraph. In accordance with this, the output shows that a “Next” link is inserted after the first paragraph. The next page displays the remaining content.

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//MOBILEAWARE//DTD MMXHTML 1.2//EN"
"http://www.mobileaware.com/DTD/mmxhtml_1.2.dtd">
<html>
<head>
    <title>Soccer News</title>
</head>
<mm-structure id="structure-1" where="IsMenuDriven" >
    <mm-group-ref idref="group1" type="normal" depth="0" display="all"/>
</mm-structure>
<body>
  <mm-group title="group1" id="group1" >
    <mm-head useradded="yes" id="group1_head" >Group1 head text</mm-head>
    <mm-body id="group1_body" idref="group1_head" >
        <p style="page-break-after: suggested" >
        Uefa says Manchester United's Champions League clash with Juventus will
        go ahead.
        </p>
        Juve (probable): Chimenti; Thuram, Ferrara, Montero, Pessotto;
        Conte,…
        <br/>
    </mm-body>
  </mm-group>
</body>
</html>
```

Increasing Content Readability



Make Your Site Easy to Navigate

Presenting the user with a list of links to the content makes your site more navigable and easier to use.

As you organize content into groups and sub-groups, each with its own head and body, you are effectively creating a hierarchical arrangement of content.

WebLogic Mobility Server makes it possible to present this hierarchy as a set of navigation links. It generates these links automatically by compiling a list based on the headings you assigned to groups as you organized your content.

It works like a collapsible outline: You can set the number of heading levels to be shown and specify whether any links contained in the body get shown as well.

For HTML devices, it means you can reference a group on a layout so that it only extracts and displays the links within the group. On menu-driven devices, it means you can create a navigation hierarchy to the groups on your request page.

Create a Navigation Hierarchy

When you use WebLogic Mobility Server to generate a navigation hierarchy, you can control how many levels are shown. You can choose to just show links to the parent groups or provide additional links to their sub-groups as well.

Create a Navigation Hierarchy for Menu-Driven Devices

Earlier in "Organize Content for Handheld Devices", we described the use of layouts and structures in WebLogic Mobility Server that involves the use of `<mm-group-ref>` within an `<mm-structure>` tag. The navigation that WebLogic Mobility Server generates for menu-driven devices requires a more detailed discussion of this tag.

The format of the `<mm-group-ref>` tag is:

```
<mm-group-ref idref="..." depth=".." display="…" type="…"/>
```

to generate this list, where `idref` is the unique ID of the group you want to reference and depth is number of levels you want in the hierarchy. The `type` attribute should be set to "normal" and `display` set to "headings." The `depth` attribute is set depending on the desired effect on the generated Navigation Menu.

Depth Attribute Effect on Navigation Menu

Depth	Effect on generated Navigation Menu
0	Displays a link to the parent group
1	Displays a collection of links to the parent group and the immediate child groups
2	Displays a collection of links to the parent and all nested sub-groups.

Use Options Menus

Some phones have the capability to display links in an options menu that can be called up from within any WML page. These links can be used to provide a handy way to bring users to various parts of a site.

For example, a page may be split into five cards on a deck. Assume the user is browsing the last card and wishes to navigate back to the first card (back to the top of the page). This task can be made easier by the author providing a link to Home in the options menu rather than the user having to click **Back** four times.

To create an option menu you use the XHTML `<meta>` tag. The mmXHTML syntax is as follows:

```
<meta name="..." content="..." scheme="mmsection"/>
```

The WebLogic Mobility Server JSP taglib syntax:

```
<mm:meta name="..." content="..." scheme="mmsection"/>
```

Attributes

- **name**

  Specifies text that will appear as the link in the options menu.

- **Content**

  Relative URL that the link should go to.

- **Scheme**

  An identifier for WebLogic Mobility Server, the value should always be "mmsection".

The normal XHTML rules applies, that is,, the `<meta>` or `<mm:meta>` tag must be located within the `<head>` element.

Note: The Options menu will not be delivered to XHTML MP devices. This feature only works on WML devices that support the Options menu.

Back to Top

WebLogic Mobility Server can be configured to automatically insert navigation shortcuts by setting the following parameter in the *mis.properties* file:

```
backtotop.enabled: true
```

`This` configuration entry defines whether the “Back to Top” feature is enabled or disabled.

Note: Back to top functionality applies to menu-driven devices only. It is disabled by default.

When back to top is enabled a shortcut “Back To Top” link is provided on the device, which will allow the user to return directly to the top of the group based on the hierarchy of the current document.

The text used when inserting the “Back to Top” link can be configured in the *contentassembly.properties* file as described in the *BEA WebLogic Mobility Server Administration Guide*.

Back to Top examples

The following example analysis describes navigation of the user from the highest level in the current document hierarchy, the navigation page, to the lowest level in the current document hierarchy where content has been split across multiple screens.

Each example describes the functionality of the Back To Top link, which is dependent on the use cases described here.

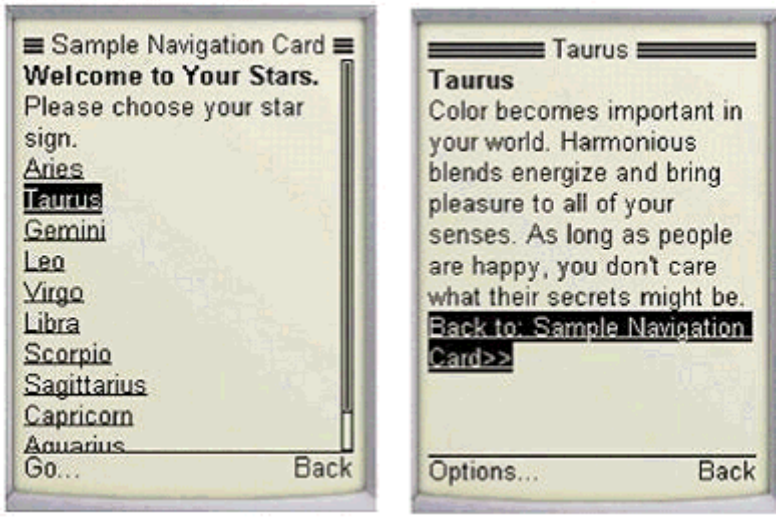
Example 1: Select “Back To Top” from a group immediately below the navigation page

This example describes the situation where the user has navigated to a point in the current document that is one level below the navigation page.

Here usage of the Back To Top link will bring the user directly to the navigation page. This is shown in the following graphic.

Each star sign content page will have a link back to the navigation page if back to top is enabled.

Content Page With Link Back To Navigation Page



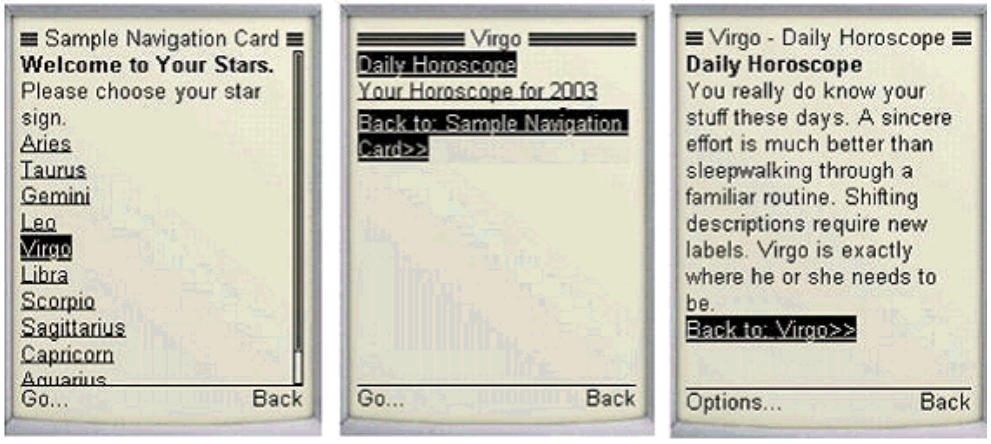
Example 2: Select “Back To Top” from the top page in a subgroup

This example describes the situation where the user has navigated to a point in the current document that is two or more levels of content below the navigation page. In this case, Virgo has a three level page structure, including an intermediate page between the navigation page and the daily horoscope.

Here usage of the Back To Top link will bring the user one level up the document hierarchy. In the following example, the user is at the Virgo - Daily Horoscope page. Usage of the Back To Top link will bring the user one level higher up the document hierarchy. In this example it brings the user from the “Virgo - Daily Horoscope” content to the top of the “Virgo” page, which is one level higher in the current document hierarchy.

Continuous usage of the Back to Top link will eventually bring the user to the navigation page for this document, that is, “Welcome to Your Stars”.

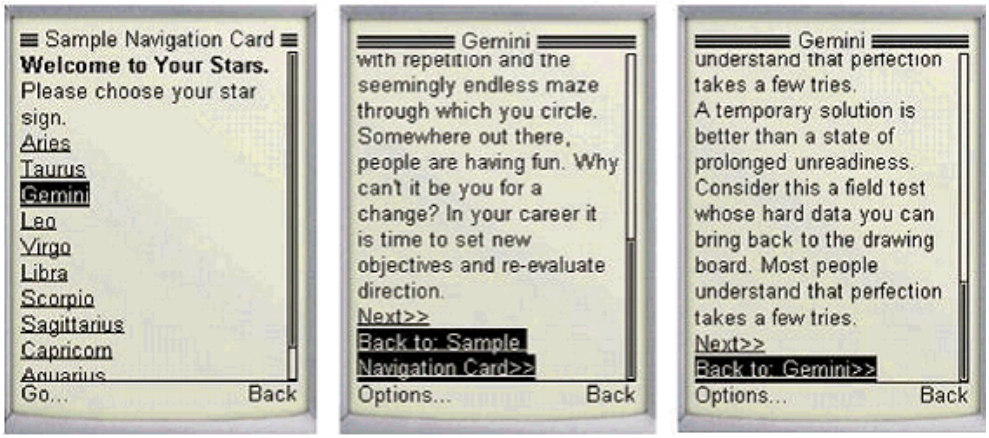
Selecting “Back To Top” From Top Page In Subgroup



Example 3: Select “Back To Top” from a “Next” page on a split group

This example describes the situation where the user has navigated to the second or subsequent pages of content for this group. This is shown in the following second and third graphics.

Selecting "Back To Top"|From “Next” Page



Here WebLogic Mobility Server will automatically inline a shortcut underneath the **Next>>** link. Via this link, the user will be able to navigate directly to the top of the group without having to repeatedly press the “Back” option.

Work with Tables

This section introduces the <mm-table-model> tag that gives content authors control over how tables are delivered to menu-driven devices and PDAs.

Note: The examples in this section use the JSP taglib mobility tags. They will work equally well using mmXHTML.

The Problem with Tables

Tables are useful for presenting complex data in a readable format. They are also used as layout tools for designers wanting to arrange text and graphical elements on a web page. Tables, however, pose a problem for many mobile devices. For example:

- Tables are slow to download
- It is not uncommon for tables to render differently on different devices
- Because of smaller screens, tables will often be wider than the viewable width of the device
- Some WML devices do not support tables at all

Table Meaning Lost on Device Without Table Support

Table Support but Small Screen

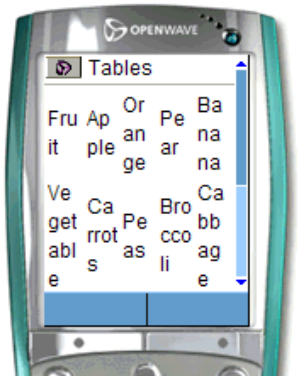


Image Courtesy of Openwave Systems Inc

The <mm:table-model> Tag

WebLogic Mobility Server addresses the problem with the `<mm:table-model>` element. This tag allows authors to tailor table information to the requesting device so that the information is displayed in a readable fashion regardless of the device's capabilities. It can deliver all or part of a table, flatten tables for devices that cannot render them, and give authors the power to create multiple configurations for the different requesting device types.

The table-model tag is a self-closing tag, which means that instead of a corresponding `</mm:table-model>`, it has a slash before the final angle bracket as can be seen here:

```
mm:table-model major="row" headlocation="1" bodylocation="*" tabletype="normal"
…/>
```

This tag is usually placed directly after the XHTML `<table>` tag. Nested tables, if any, need their own `<mm:table-model>` tag.

Note: This tag is specifically for controlling the display of tables on handheld devices. The functionality of `<mm:table-model>` has no effect on PC browsers.

Understand the Table-Model attributes

This section will cover the various attributes of `<mm:table-model>` and describe the effect that they have when delivering tables to handheld devices.

It is important to understand how WebLogic Mobility Server interprets the difference between a row-major and a column-major table.

Attributes

The `major` and `headlocation` attributes work together to tell WebLogic Mobility Server about the orientation of the table data.

- **major**="**row | column**" (required attribute)
- **headlocation**="**…**" (required attribute)

  The following table shows a column-major table with headlocation="1". This means that the headings that are located in column 1 should be used to orient the transformed table.

  Headings in Column 1 (major="column" headlocation="1")

Fruit	Apple	Orange	Pear	Banana
Vegetable	Carrots	Peas	Broccoli	Cabbage
Meat	Beef	Lamb	Chicken	Fish
Dairy	Milk	Yoghurt	Butter	Cheese

  The following table shows a row-major table with headlocation="1". As a row-major table, the headings are located in row 1.

Headings in Row 1 (major="row" headlocation="1")

Fruit	Vegetable	Meat	Dairy
Apple	Carrots	Beef	Milk
Orange	Peas	Lamb	Yoghurt
Pear	Broccoli	Chicken	Butter
Banana	Cabbage	Fish	Cheese

This orientation is important if using the attribute `bodylocation` to display only a portion of the table. WebLogic Mobility Server will understand which way to split the table so that meaning is not lost.

- **bodylocation**="**…**" (optional attribute)

  This attribute specifies which rows or columns are to be displayed. To display the entire table the value should be set to "*". Without this attribute explicitly defined, WebLogic Mobility Server will default to displaying the entire table although a warning will appear in the WebLogic Mobility Server Application Server console. A portion of the table can be displayed by setting this attribute to a space-separated list of numbers representing the rows (or columns) to be sent to the device. Using the preceding table as an example, the author might wish to see only vegetarian items in the table. To do this, set `bodylocation="1 2 4"`. For vegans, set `bodylocation="1 2"`.

  Table with major="row" bodylocation="1 2"

- **sdtransform** (optional attribute)
  - sdtransform="base-transform"

    This is the attribute used to flatten tables. If a table is delivered to a non-table supporting device, the content is presented sequentially and often, all meaning is lost). Even if a mobile phone or PDA can process tables, often the screen dimensions will cause the table information to wrap excessively or to revert to a linear display if the table is still too wide. If an <mm:table-model> tag is placed within the table with the sdtransform

attribute set to “base-transform” the table data is extracted from the rows (or columns) and paired with the data from the headlocation in a “table header: table data” pairing.

Table Flattened: sdtransform="base transform"

Image Courtesy of Openwave Systems Inc

- sdtransform="on-same-card"

  This is used to display new tables on the same card.

  Note: This will not create a link to that nested table

- sdtransform="on-new-card"

  This is used to create a link to a nested table that is placed on a separate card

- **tabletype**="normal | group" (required attribute)

  If the attribute `tabletype` is set to "normal", WebLogic Mobility Server will attempt to display the entire table. If the table is bigger than what can fit onto a “card”, the page will be broken into multiple cards.
  Setting `tabletype` to “group” causes the table headers to be rendered as links. These links can be navigated to view detailed table content presented in a flattened "table header: table data" pairing.
  Using `tabletype="group"` for a device that can render tables will have little effect unless the table is first “flattened” using `sdtransform`.

  The following graphics illustrate the use of tabletype="group", where the table has been rendered with the headers as links to “flattened” tables

  Table with tabletype="group" - Entry Link

Image Courtesy of Openwave Systems Inc

Table with tabletype="group" – Header Links

Image Courtesy of Openwave Systems Inc

Table with tabletype="group" - Sub-table View

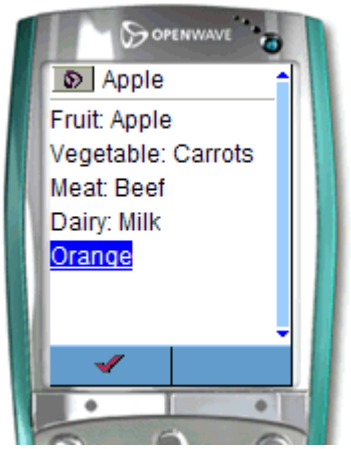


Image Courtesy of Openwave Systems Inc

- **where**="**…**" (optional attribute)

  The table-model tag uses the where attribute to specify a particular table configuration for a specific device or group of devices. Authors can use multiple table-model tags for each table defined in their content. The where attribute in the table-model tag might be used to deliver a table to devices that support tables, but to flatten the table information for devices that do not support tables as can be seen in the following content segment.

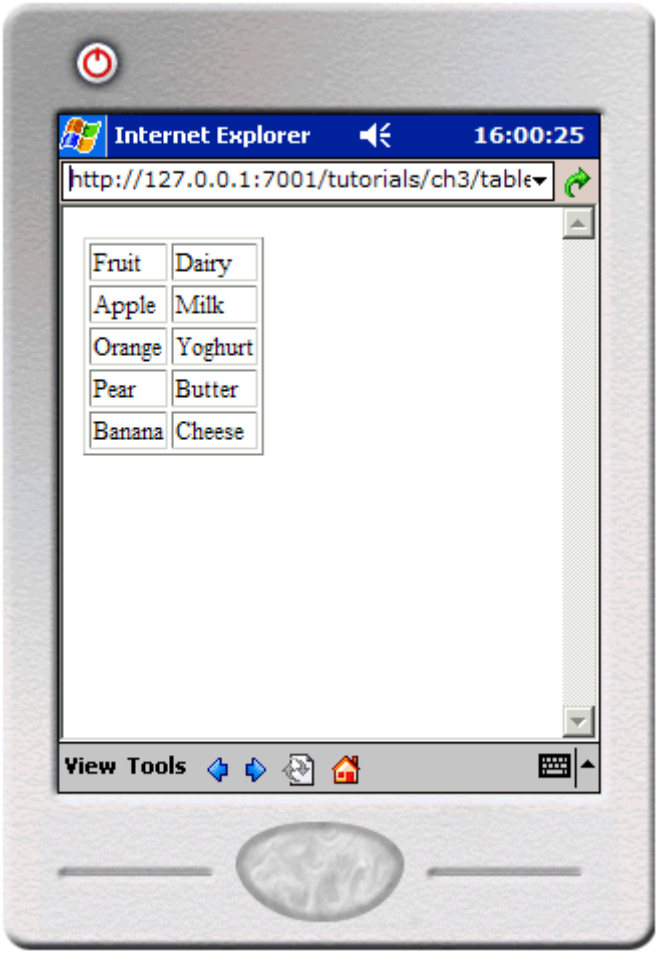
```
<table border="1">
<mm:table-model major="row" headlocation="1" bodylocation="1 4"
 tabletype="normal" where="UAProf.BrowserUA.TablesCapable" />
<mm:table-model major="row" headlocation="1" bodylocation="1 4"
 tabletype="normal" sdtransform="base-transform" where=" not
 UAProf.BrowserUA.TablesCapable"/>
<tr><td>Fruit</td>
    <td>Vegetables</td>
    <td>Meat</td><td>Dairy</td></tr>
<tr><td>Apple</td>
    <td>Carrots</td>
    <td>Beef</td>
    <td>Milk</td></tr>
…
```

```
</table>
```

Note: The use of where="IsFullBrowser" is not allowed.

WebLogic Mobility Server identifies which devices do not support tables and “flattens” the data being delivered to that device.

Delivery to Device Supporting Tables



Flattened Table

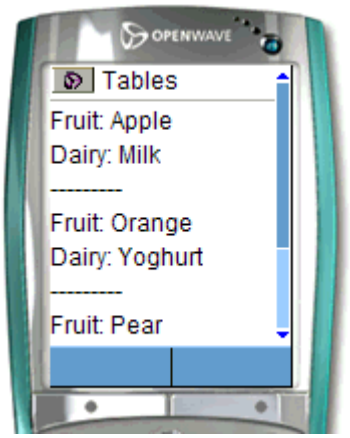


Image Courtesy of Openwave Systems Inc

Note: When multiple table-model tags are present, WebLogic Mobility Server uses the *first* one who's where clause conditions match the requesting device.

An example of a more complex where condition can be seen in the following code block where the table will be flattened even if the requesting device supports tables if the screen width is less that 200 pixels wide. This would stop excessive wrapping of table data as the device tries to render the table on the small screen.

```
<mm:table-model major="column" headlocation="1" bodylocation="*"
tabletype="normal" sdtransform="base-transform"
where="UAProf.BrowserUA.TablesCapable and UsableWidthPixels < 200"/>
```

title="…" (optional attribute)

If present, the value of this attribute forms the table title when used with the tabletype="group" as can be seen in the preceding example where the title in the first frame in is "My Table". If title is not present when tabletype="group" and a link name is required, the keyword "Data" will be used instead.

Tables and Style Sheets

External style sheets must be used to style tables for handheld devices. For example, the following table attributes will be ignored:

```
<table border="1" cellspacing="0">
```

Instead, place a table class in an external style sheet:

mystyles.css

```
table.myclass {
text-align:center;
border-style: solid;
border-width: thin;
border-collapse: collape;
}
```

Reference this style from within the <table> tag:

```
<table class="myclass">
```

And remember to link the style sheet to the document containing the table.

```
<html>
<head>
   <title>Tables</title>
   <link rel="stylesheet" href="mystyles.css" type="text/css"/>
</head>
```

Work with Images

This section describes the techniques for delivering images to different devices. It explains how to provide alternative objects so they match the display capabilities of the target device.

The section introduces the following tags:

- `<mm-img>`
- `<mm-media-group>`
- `<mm-logo>`

About Images

Devices differ in how they handle and present graphical elements. PC browsers can display image formats such as GIF and JPEG, whereas some WML devices are restricted to the WBMP format. WebLogic Mobility Server provides a set of tags that manage graphic delivery to a variety of client devices. These tags allow you to specify the correct image format for the requesting device type.

For example, you could specify a 200 x 100 pixel GIF for a PC-browser, a 50 X 25 pixel GIF for a PDA and a WBMP image for a mobile phone that can only support the WBMP format. When WebLogic Mobility Server is transforming the content, it will select the image that best suits the requesting browser.

Use <mm-img>

Send the Correct Image Type

The `<mm-img>` tag can be used to place images in content that is intended for a variety of different client devices. This tag supports all the attributes of the XHTML `<img>` tag with the additional functionality of the where attribute which allows you to target the device for which the image is intended.

Note: This is an empty tag, so remember to place a slash (/) before the final angle bracket (>) to keep your content “well-formed”.

To handle a GIF image, you could use the following syntax to ensure that the image would not be sent to a WML browser that could not handle this particular image format.

```
<mm-img src="dog.gif" alt="Dog" height="54" width=80" where="ImgGIFSupported"/>
```

Resize the Image to Fit the Device Screen

Because handheld devices have widely varying screen sizes, you sometimes need to be able to dynamically manipulate the size of the image depending on the requesting device type. The `<mm-img>` tag uses the `fittoscreen` attribute to give more control over how an image displays on a small screen.

If set to true, the image width is resized to the `UsableWidthPixels` value of the device as defined in the Device Repository. The image height is resized by the same factor so that the image will maintain the same aspect ratio.

Note: This attribute only has an effect if the original image is wider than the targeted screen width.

Often when developing multi-channel content, you will have several images of different formats and sizes that would be appropriate for each device or device class targeted. You could put each <mm-img> tag inside an <mm-include> tag or instead put them all inside an <mm-media-group> tag so that the best image available will be selected for a particular device class.

Use <mm-media-group>

WebLogic Mobility Server provides the tag <mm-media-group> that can hold several different image tags. An <mm-img> tag represents each image. When WebLogic Mobility Server is transforming the content, it will select the image that best suits the requesting device.

```
<mm-media-group alt="No image available">
   <mm-img where="IsFullBrowser" src="greatDane.gif" height="120" width="80"
alt="Great Dane"/>
   <mm-img where="ImgGIFSupported" src="terrier.gif" height="30" width="30"
alt="Terrier"/>
   <mm-img where="ImgWBMPSupported" src="sm_dog.wbmp" height="30" width="30"
alt="Chihuahua"/>
</mm-media-group>
```

WebLogic Mobility Server will select the first image that satisfies the where clause. In the preceding example, any full browser will match the first <mm-img> condition and will receive a picture of a Great Dane. Menu-driven devices or PDAs that can accept GIF images will receive a picture of a terrier. Finally, handheld devices that cannot accept GIFs, but do understand the WBMP format will be sent a picture of a Chihuahua. A device that does not meet any of these conditions will display the media group's alt tag of "No image available".

Display a Logo on a WML Device

A logo is a graphic symbol designed to reflect the corporate or product identity so it is instantly recognizable by consumers. It is sometimes the first item to appear when a user visits a service.

On WML devices, logos are treated as a special case, appearing for a short interval (usually about 5 seconds) and then being replaced with the main page.

To take advantage of this feature, wrap an image within the <mm-logo> element so that it gets treated differently on a WML device. WebLogic Mobility Server supports any image type with <mm-logo>. If the device is WML, the image is delivered as a WAP logo, otherwise, <mm-logo>, along with its contents, is removed.

Define a Logo

Use the following tag to define a logo:

```
<mm-logo id="name" displaymode="once" period="2">
```

where “name” is unique identifier for your logo.

Within this tag, insert the code for the image:

```
<img src="image.location"/>
```

where “image.location” is the location on the server of the image.

Complete the element with the closing tag:

```
</mm-logo>
```

Set the Displaymode and Period Attributes

Example 1: The author wants a logo displayed “once per session” and for a period of two seconds.

The author indicates that this logo should be displayed once per session by adding the attribute `displaymode="once"` to the `<mm-logo>` tag. The display period of the logo is indicated by adding the attribute period="2", where the integer represents the number of seconds that the logo will be displayed.

```
<mm-logo displaymode="once" period="2">
    <img src="img/logo.wbmp" alt="logo"/>
</mm-logo>
```

Example 2: The author wants a logo displayed always and for a period of three seconds.

The author indicates that this logo should be displayed always (each time the page is requested) by adding the attribute `displaymode="always"` to the `<mm-logo>` tag. The display period of the logo is indicated by adding the attribute `period="3"`, where the integer represents the number of seconds that the logo will be displayed for.

```
<mm-logo displaymode="always" period="3">
    <img src="img/logo.wbmp" alt="logo"/>
</mm-logo>
```

Deliver Different Image Formats with <mm:logo>

Several WAP devices support the display of color images such as GIFs. Through use of the `<mm:media-group>` and `<mm:img>` tags, you can enhance the functionality of `<mm:logo>` to deliver color images to these devices whilst delivering monochrome images to others. For instance:

```
<mm:logo displaymode="once" period="3">
    <mm:media-group alt="">
      <mm:img where="ImgGIFSupported" src="myimage.gif" alt="Logo"/>
      <mm:img where="ImgWBMPSupported" src="myimage.wbmp" alt="Logo"/>
    </mm:media-group>
  </mm:logo>
```

Note: You must include the alt attribute on each `<mm:img>` tag, and you must specify `<mm:img where="ImgWBMPSupported" ...>` as the final image in the list.

Work with Character Sets

Character encoding is an algorithmic process that specifies how human-readable characters are converted into bytes for storage or transmission. Characters in a language (or set of languages) are mapped to numbers represented by bytes (or octets). Character decoding is the process of converting bytes into characters.

To avoid encoding errors during the process of storing, transmitting and displaying a document on the web, a single consistent method of encoding / decoding should be used throughout.

This document explains how to avoid and resolve encoding problems.

About Character Encoding/Decoding

Character encoding is a method of converting characters into bytes and decoding is a method of converting bytes into characters.

The standard character set for computers has traditionally been ASCII (American Standard Code for Information Interchange). No provision is made in ASCII for foreign characters or specialized symbols. Hence, various so-called "extended ASCII" sets have been developed to provide these symbols. However, the Web has adopted an extended character set, ISO 8859-1 (otherwise known as ISO Latin-1), as its standard.

In addition, to avoid a preference for one language over another, HTML 4.0 has adopted Unicode as its official document character set. Unicode is attempting to create a single character set under which every character, from every language in every region can be represented.

Encode Mechanisms

An application must select a character encoding / decoding method when it is opening, validating or displaying a HTML document. For documents in English and most other Western European languages, the character encoding ISO-8859-1 is typically used.

There are a number of mechanisms within the HTTP, XML and HTML protocols for specifying the character encoding:

- Unicode encoded-documents commonly use Byte Order Marks (BOM) to inform the decoding software which algorithm needs to be used to decode the byte stream correctly. This is simply a set of defined lead bytes that mark the stream as being of a particular type.
- The HTTP protocol defines a response header called “Content-Type” which can include the character set name as part of its value (when the mime-type is text/*). The HTTP server needs to be configured to set this header. For example, to specify that an HTML document uses ISO-8859-1, a server would send the following header:

  ```
  Content-Type: text/html; charset=ISO-8859-1
  ```

  In XML, the XML declaration can contain the document encoding:

  ```
  <?xml version="1.0" encoding="ISO-8859-1"?>
  ```

- In HTML, a <meta> tag can be used to define the document encoding

  ```
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
  ```

How Content Gets Encoded

Content is encoded/decoded by the content editor (such as a text editor or database), server, JSP file and display device.

Errors will occur if the same encoding / decoding algorithm is not consistently used along the supply chain from creating content to delivering it to the end device.

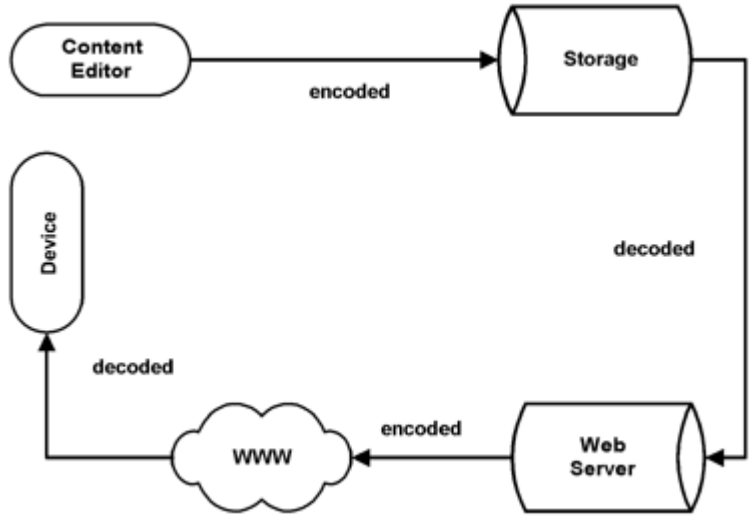
Text Editors and Databases

Character encoding begins when a file is created and stored. Text editors such as Microsoft Notepad, Macromedia Dreamweaver, Adobe GoLive, CoffeeCup store the content with a default encoding. When writing data to a database, a servlet must use the same encoding as the data stored in the database.

Usually, you can specify the default encoding to be used when saving and opening files. For example, in Macromedia Dreamweaver, select the Preferences | Fonts | Encoding option and specify the Default encoding to be used. For example, Western (Latin 1) or Japanese (Shift JIS).

Each component in the content supply chain is responsible for encoding / decoding the content and for passing on the Content-Type header (if present) to the next component in the chain.

Content Supply Chain



Servers

The preferred method of indicating the encoding is by using the charset parameter of the Content-Type HTTP header. For example, to specify that an HTML document uses ISO-8859-1, a server would send the following header:

```
Content-Type: text/html; charset=ISO-8859-1
```

Configuring an application server to use a particular encoding depends on the individual server. For example, if you are using the Apache server, you can add a file named *.htaccess* to any directory to set the Content-Type of files in that directory and any sub-directories.

JSP Files and Servlets

To ensure the output of your JSP page matches the encoding of the rest of the system, insert the following encoding statement in your JSP file:

```
<%@ page contentType="encoding" %>
```

where encoding is the encoding of your choice.

For servlets, you can specify content type and character encoding in your HTTPServletRequest and HTTPServletResponse objects using:

- setCharacterEncoding() in request objects and
- setContentType() in response objects.

For example:

```
HttpServletResponse.setContentType("text/html;charset=iso-8859-1");
```

How WebLogic Mobility Server Determines Character Encoding

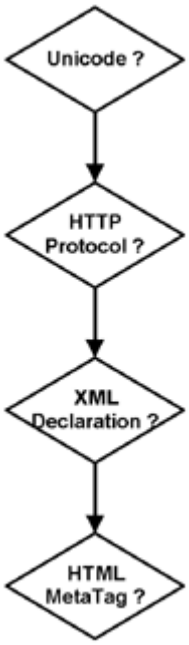
While a web server may pass on the byte stream without decoding, WebLogic Mobility Server needs to decode / encode the mark-up so it can perform the appropriate transformations and then recode the result for transmission.

WebLogic Mobility Server uses the following sequence to determine the character encoding for a document:

1. It determines whether Unicode is used. WebLogic Mobility Server only checks for UTF-16 Byte Order Marks so if the document is UTF-8 encoded the HTTP server must be configured to set the Content-Type header.
2. Next, WebLogic Mobility Server checks the HTTP protocol to see if the Content-type response header has been defined.
3. If not, WebLogic Mobility Server checks the XML declaration and if this doesn't exist, the HTML meta tag is checked.

   ```
   <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
   ```

Character encoding flow



Influence the Character Encoding Delivered to the Device

WebLogic Mobility Server selects the best encoding in which to deliver transformed content to a device based, in part, on the list of preferred encodings supplied by the device.

In some circumstances, it may be desirable not to deliver content in certain of these preferred encodings – if the device is known to supply inaccurate information about its capabilities, or if no useful translation exists between the encoding of the content and an encoding preferred by the device.

An example of this is the translation of Chinese content; this cannot be represented in the ISO-8859-1 character encoding, but can be represented in the UTF-8 encoding. In these circumstances, it is preferable to ignore requests for delivery in ISO-8859-1 and to deliver in UTF-8 where understood.

Specifying a comma-separated list of encodings for the disallowedOutputEncodings property in the *mis.properties* file instructs WebLogic Mobility Server never to deliver content in any of these encodings.

Example

```
disallowedOutputEncodings: iso-8859-1, iso-8859-5
```

This example instructs WebLogic Mobility Server never to deliver content in the iso-8859-1 or iso-8859-5 encodings.

Fine-Tune Mobile Content

This section describes the following features that can be used to fine-tune the presentation of mobile content:

- Horizontal rule
- Textarea
- URL compression

Horizontal Rule

For devices that do not support the horizontal rule tag, WebLogic Mobility Server always translates any `<hr/>` tags it encounters into a series of hyphen characters followed by a `<br/>` element.

The number of hyphen characters used is determined from the TextColumns device attribute stored in the Device Repository. TextColumns indicates the number of columns (characters) that the device screen can accommodate using the system font.

Notice how the horizontal rule fits the width of each screen.

Substituting <hr/> with Dashes

Emulator 1

Emulator 2

Textarea

For devices that do not support <textarea>, WebLogic Mobility Server translates any <textarea> tags it encounters into an input element of type “text”. For more information on textarea see the following URL:

http://www.w3.org/TR/html401/interact/forms.html#edef-TEXTAREA

```
<mm-group id="company_details">
   <mm-head id="hd_company_details" useradded="no">Company Details</mm-head>
   <mm-body id="bd_company_details" idref="hd_company_details">
      <p>ABC Company Ltd.</p>
      <p>North Business Park, Circular Road, Dublin, Ireland</p>
      <p>Phone: 888-000-111</p>
      <p>Fax: 000-888-111</p>
      <form action="gotothis.htm" method="get">
         <textarea name="thetext">
          First line of initial text.
          Second line of initial text.
         </textarea>
      </form>
   </mm-body>
</mm-group>
```

Use of <textarea> Tag

Use of <textarea> tag

Edit screen

Configure URL Compression

The URLs generated by portal frameworks and other content servers are often very long. If URL rewriting is used instead of cookies for session management, the length of these URLs is extended further. Because the length of these URLs takes up valuable space within the limited memory of a small device, the output visible to the user is often very limited. In extreme cases, pages are limited to just 2 or 3 links.

To mitigate this, WebLogic Mobility Server supports URL compression, which reduces the length of these URLs to a minimum, thereby allowing much more content to be delivered to the device. This is especially relevant where the device has limited memory but could also be important where limited bandwidth is an issue.

URL compression works by breaking the URL into fragments (query parameters) and replacing the fragments in the URL with shortened tokens. These shortened tokens are used by WebLogic Mobility Server to map a request generated from the replacement URL back to the original URL.

Examples

The following is an example of a URL of 359 characters produced by BEA WebLogic Portal:

```
/avitekfinancial/application?namespace=tracking&origin=searchResults.jsp&
event=link.clickContent&com.bea.event.type=com.bea.content.click.event&
com.bea.event.userid=null&com.bea.event.documentid=Avitek/DemoDocuments/Demo
Features
List.xls&com.bea.event.documenttype=AvitekDocs&contentId=Avitek/DemoDocuments/D
emo Features List.xls
```

With URL compression turned on in WebLogic Mobility Server, this URL would be reduced to 99 characters, which is a saving of 260 characters.

```
/avitekfinancial/application?2=!!3&!!4=!!5&!!6=!!7&!!8=!!9&!!10=!!11&!!12=!!13&
!!14=!!15&!!16=!!13
```

URL compression can be configured in the *mis.properties* file. The following is a sample configuration for WebLogic Mobility Server running against a BEA WebLogic Portal server.

URL Compression Configuration for BEA WebLogic Portal

Property	Description
url.compression.store.type	Defines the store type to be used. The only valid type in WebLogic Mobility Server is session. Example: session
url.compression.token.prefix	The string used to prefix the compression tokens. Prefixing helps avoid clashes with uncompressed tokens that may have the same value as a compressed token. Default is "!!". (Note: The Nokia Mobile Internet Toolkit 3.1 does not support "!!")
url.compression.params	Comma separated list of query parameter names to be compressed. Example: namespace, event, com.bea.event.type, com.bea.event.userid, com.bea.event.documentid, com.bea.event.documenttype, contentId, origin, pageid, portletid
url.compression.vals	Comma separated list of query parameter names for which values can be compressed. Elements in this list must be members of the url.compression.params list. Example: namespace, event, com.bea.event.type, com.bea.event.userid, com.bea.event.documentid, com.bea.event.documenttype, contentId, origin, pageid, portletid
url.compression.fail.redirect	Webapp-relative redirect URL for failed decompression. When WebLogic Mobility Server receives one or more expired compression tokens in a request it redirects to this URL. This typically occurs after the session has timed out. for example /avitekfinancial/application

Important note: When using the redirect URL for failed decompression it is recommended that content developers design JSP or XHTML pages that do not make use of, or depend on, the values of parameters passed in the URL.

iMode Support

iMode is NTT DoCoMo's popular mobile internet access system which started in February 1999 in Japan. iMode has adopted cHTML (compact HTML) as its mark-up language. Compact HTML is a subset of HTML dealing primarily with text and simple graphics.

WebLogic Mobility Server supports this emerging technology. Content for iMode devices should be structured according to the general authoring guidelines for small devices as described in this manual. There are, however, a few iMode-specific methods of mobilizing your content. This section describes these practices.

Conditional Content

In order to include or exclude specific content for an iMode device, you can use the database attribute DeliveringIHTML. For example:

```
<mm:include where="DeliveringIHTML">
     &#59677;
</mm:include>
```

In most cases, it is not necessary to be this granular. Using IsMenuDriven should suffice.

Access Key Support

WebLogic Mobility Server supports the use of access keys on iMode phones. They work in virtually the same way as other devices that support access keys EXCEPT the attributes `assignall` and `assignempty` are not supported. WebLogic Mobility Server inserts the appropriate emoji characters representing the access key number next to the link.

Emoji Characters

Emoji characters are the 12 x 12 pixel picture characters used on iMode phones. They are like emoticons but are smaller and therefore easier to transmit. WebLogic Mobility Server fully supports these characters. To add an emoji icon, simply place an iMode conditional expression with the emoji character (as represented by its Unicode or SJIS equivalent) into your content. For example:

```
<mm:include where="DeliveringIHTML">
   &#59677;
</mm:include>
```

iMode-Specific Styles

Generally, styling that is created for menu-driven devices will also work well on iMode devices. As an author, however, you may wish to take advantage of specific capabilities of an iMode device. To do this, it is recommended that you use an external CSS style sheet that contains styles exclusively for iMode devices. You would create a style sheet with iMode-specific styles using the `<link>` and `<mm:include>` elements, making it accessible only to iModes. For example:

```
<mm:include where="DeliveringIHTML">
     <link href="iModeStyles.css" rel="stylesheet" type="text/css">
</mm:include>
```

Marquee

Marquee is supported using CSS syntax.

To create text that scrolls across the screen, you might have the following in your content.

```
<div class="marquee-styleA>
     This is my scrolling text!
</div>
```

Your style sheet would define the marquee style.

```
div.marquee-styleA {
     marquee-style: slide;
     marquee-direction: left;
     marquee-repetition: 1
}
```

Blink

Similar to marquee, blink is supported through the use of a style sheet.

```
<div class="blink-style">
     Phone me!!
</div>
```

Your style sheet would define the style as follows:

```
div.blink-style {
     text-decoration: blink;}
```

istyle Support

The `istyle` attribute is also supported using CSS. Currently iMode uses this attribute with the input type="text" and “textarea” elements. This attribute indicates the input mode for the phone.

The possible values that are supported are as follows:

Supported istyle values

3	Alphanumeric: Pressing the 2 repeatedly yields: a b c 2 …
4	Numeric: Pressing the 2 repeatedly yields: 2222

For example, your code might have the line:

```
<input type="text" id="idA" name-"inputA"/>
```

Your iMode style sheet would then have the following style defined:

```
input#idA {
   istyle: 3;
}
```

This would apply the alphanumeric input mode to your textbox.

Phone Number Dialing and CTI

WebLogic Mobility Server supports the phone number dialing capabilities of iMode devices. The mobility tag `<mm:phone-number>` has a new `cti` attribute which mimics the cHTML attribute `cti` which was added in cHTML 2.0. This attribute allows you to create a link that will be dialed when the user selects the link. Unlike the `num` attribute of `<mm:phone-number>`, `cti` gives the user the option of including pauses, characters and extension numbers after the main phone number. The hash mark (#) and the asterisk (*) are also supported.

Supported Digits and Characters in CTI Attribute

Digit / Character	Description
0 - 9	These digits will be transmitted.
* #	These tones will be transmitted.
,	Wait for 1 second.
/	Pause. Placing this in the ***cti*** string will cause a pause to wait for user input.

```
<mm:phone-number num="+ 35312410500" cti="+35312410500/,,538#">
     Call Julia.
</mm:phone-number>
```

The previous example snippet will result in the following:

- The phone will dial +35312410500.
- It will wait for user key input once the receiving end picks up.
- After key input, it will wait 2 seconds before dialing the extension "538#".

Phones that do not support the `cti` attribute will use the value of the `num` attribute and dial +35312410500.

Part V The Delivery Context API

Overview of the CC/PP Delivery Context API

The WebLogic Mobility Server transformation engine regularly checks information about the requesting device in order to translate content into an appropriate language and format that the device can understand. There may be times, however, that a content developer requires specific information about a device for the purpose of fine tuning the layout or styling of their web content. This section demonstrates how to access this device information from the Device Repository using the JSR188 API.

About the Device Repository

Networked, mobile and wireless devices vary widely in their ability to support different aspects of web content. Such characteristics as screen size, image and color capabilities, script support and deck size can all affect the type of content that can be delivered to them.

WebLogic Mobility Server accommodates these differences by maintaining a Device Repository; a datastore that contains profiles describing the properties and capabilities of a range of devices on the market. The data, stored in attribute-value pairs, enables WebLogic Mobility Server to identify the requesting device so that the content can be transformed appropriately before being delivered. In this way, WebLogic Mobility Server can tailor the presentation and delivery to each client device. Each device profile, or set of attributes defining the presentation and delivery capabilities of a device, is known as the delivery context.

When WebLogic Mobility Server receives an end-user device request, it identifies the device using a combination of incoming request header information (which indicates the mark-up language of the device and often provides device model information) and stored device attributes.

Device Profiles

To facilitate the development of device independent applications, the W3C has recently defined a standard known as Composite Capabilities/Preferences Profile (CC/PP), which is used to describe device capabilities and user preferences (that is, the delivery context). Based on this standard, the Open Mobile Alliance, the group that establishes open global standards for the mobile community has defined their own standard known as User Agent Profile (UAProf). This new standard has been adopted for the Device Repository. Currently, the database is CC/PP compliant, containing both the UAProf attribute set and a more comprehensive set of WebLogic Mobility Server proprietary device properties.

In the Device Repository, the CC/PP compliant attribute names begin with one of seven prefixes:

- UAProf.BrowserUA
- UAProf.HardwarePlatform
- UAProg.MmsCharacteristics
- UAProf.NetworkCharacteristics
- UAProf.PushCharacteristics
- UAProf.SoftwarePlatform
- UAProf.WapCharacteristics

The proprietary device attributes have no prefix.

Some examples of these attributes are:

- J2MESupported
- IsMenuDriven
- ImgWBMPSupported

For further information on the full set of attributes, see the *BEA WebLogic Mobility Server Administration Guide*.

Access CC/PP Device Profile Information

CC/PP and UAProf Attributes

JSR188 is a standard set of APIs developed by the Java Community to access delivery context information. It is these methods that developers can use to query the WebLogic Mobility Server database to gain access to the CC/PP delivery context information.

The following example demonstrates how to use the API to find out the screen size of a device, the image formats it supports and whether it supports tables.

Each of these attributes is a UAProf attribute. In the Device Repository, they are listed with their full prefix:

- UAProf.HardwarePlatform.ScreenSize
- UAProf.SoftwarePlatform.CcppAccept
- UAProf.BrowserUA.TablesCapable

These attributes do not require the prefix here. By importing the javax.ccpp.* package, the prefix is understood.

Note: If using the CC/PP attributes to specify a device or set of devices in a tag that uses the `where` attribute, you must use the full prefix. For example:

```
<mm:include where="UAProf.HardwarePlatform.ScreenSize > 180">
```

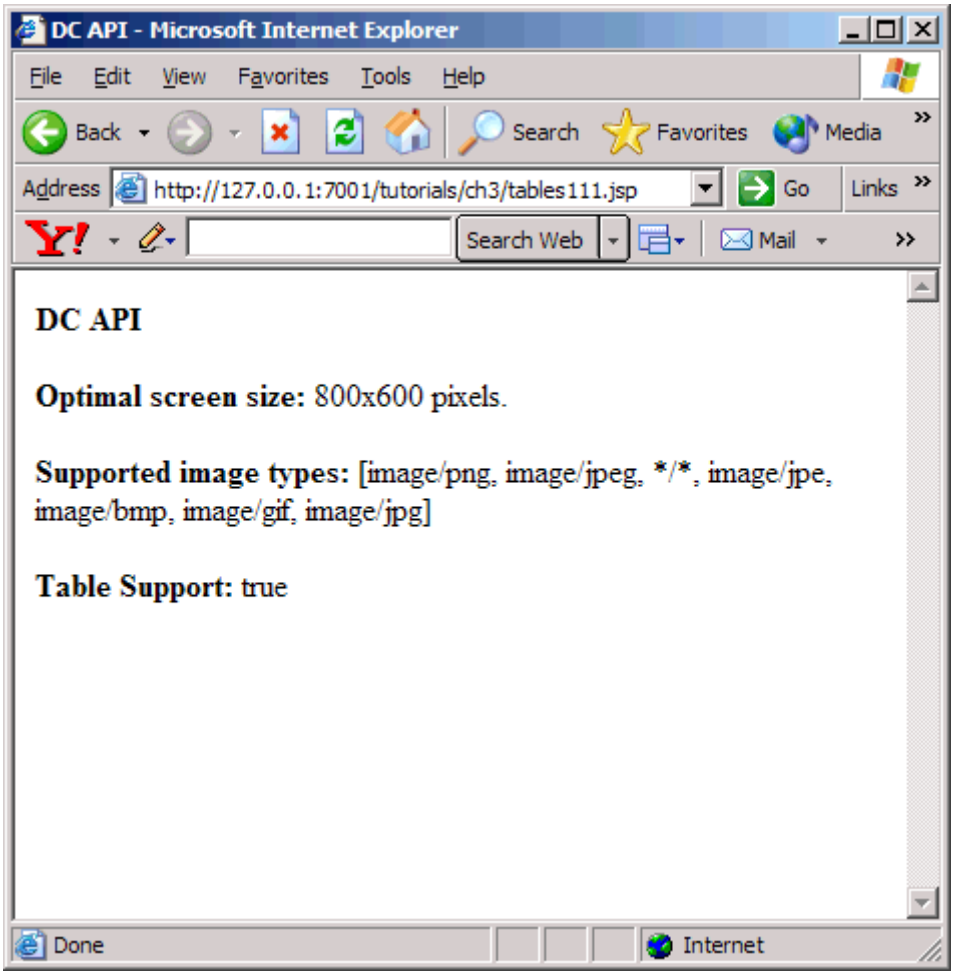
Delivery Context Example

Example: Obtain device information using the JSR188 API

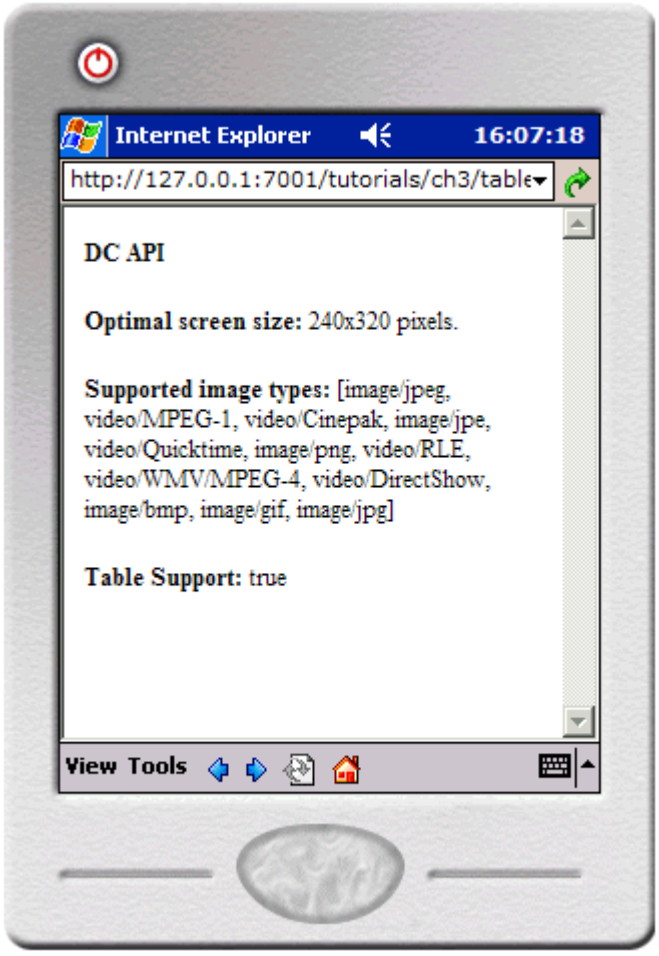
```
<%@ page contentType="text/html; charset=iso-8859-1" import="javax.ccpp.*" %>
<%@ taglib uri="mmJSPtaglib" prefix="mm"%>
<mm:page>
<%
Attribute screenSize = null;
Attribute imgFormats = null;
Attribute tables = null;
String screenSizeStr = "";
String imgFormatsStr = "";
String tablesStr = "";

ProfileFactory pf = ProfileFactory.getInstance();
Profile myProf = null;
if (pf == null) {
  System.out.println("Cannot create ProfileFactory instance.");
}
else {
myProf = pf.newProfile(request);
screenSize = myProf.getAttribute("ScreenSize");
imgFormats = myProf.getAttribute("CcppAccept");
tables = myProf.getAttribute("TablesCapable");
screenSizeStr = screenSize.getValue().toString();
imgFormatsStr = imgFormats.getValue().toString();
tablesStr = tables.getValue().toString();
}
%>
<html>
<head>  <title>DC API</title></head>
<mm:structure id="str1" where="IsMenuDriven"
accesskeycontrol="assignempty">
  <mm:group-ref idref="gp1" depth="flat" type="normal" display="all"/>
</mm:structure>
<body>
  <mm:group id="gp1" title="API">
    <mm:head id="hd1"useradded="no"><b>DC API</b></mm:head>
    <mm:body id="bd1" idref="hd1">
      <p><strong>Optimal screen size: </strong>
      <%= screenSizeStr %> pixels.</p>
      <p><strong>Supported image types: </strong>
      <%= imgFormatsStr%></p>
      <p><strong>Table Support: </strong><%= tablesStr%></p>
    </mm:body>
  </mm:group>
</body></html>
</mm:page>
```

Results

Device Information on Desktop Browser



Device Information on Desktop Browser



Device Information on WML Device

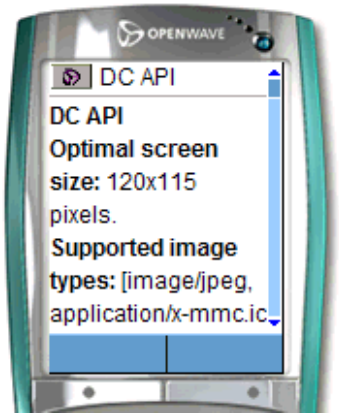


Image Courtesy of Openwave Systems Inc

Example Breakdown

In order to use the JSR188 API, you must first import the package that allows access to the methods.

```
<%@ page contentType="text/html; charset=iso-8859-1" import="javax.ccpp.*" %>
```

Initialize the variables needed for this example.

```
Attribute screenSize = null;
Attribute imgFormats = null;
Attribute tables = null;
String screenSizeStr = "";
String imgFormatsStr = "";
String tablesStr = "";
```

Create the Profile object that contains the information about the requesting device. This is done in two steps.

First, obtain an instance of a ProfileFactory object.

```
ProfileFactory pf = ProfileFactory.getInstance();
```

Then, from this ProfileFactory object, create the device Profile object.

```
myProf = pf.newProfile(request);
```

The next step is to get the required attribute from the Profile. To display the attribute value, use the method getAttribute and turn it into a string.

```
screenSize = myProf.getAttribute("ScreenSize");
imgFormats = myProf.getAttribute("CcppAccept");
tables = myProf.getAttribute("TablesCapable");
screenSizeStr = screenSize.getValue().toString();
imgFormatsStr = imgFormats.getValue().toString();
tablesStr = tables.getValue().toString();
```

The final step displays the attribute values in the HTML display that is sent to the screen of the requesting device.

```
<p><strong>Optimal screen size: </strong>
<%= screenSizeStr %> pixels.</p>
<p><strong>Supported image types: </strong>
<%= imgFormatsStr%></p>
<p><strong>Table Support: </strong><%= tablesStr%></p>
```

Note: This simple example contains an `<mm:structure>` tag for menu-driven devices. For more complex pages, this structure would be placed in a layout file, as it is considered good practice to keep the main content separate from the device-specific layout instructions.

Additional Information

JSR188 replaces the Mobility Delivery Context API. These methods are, however, still supported and must be used to access Mobility proprietary attributes. For additional details on using the Mobility Delivery Context API, see “Appendix B – Mobility Delivery Context API”.

Part VI Diagnostics

Work with Diagnostics

As you develop your multi-channel content, you will want to see how it is processed for various devices. While device emulators can show how the final output is rendered, they cannot show what is happening to the content as it is being transformed, making it difficult to diagnose and troubleshoot problems. You will need to use the WebLogic Mobility Server Diagnostic tools for this purpose.

The WebLogic Mobility Server Diagnostic tools enable developers and administrators to monitor the HTTP request / response cycle within WebLogic Mobility Server and to retrieve diagnostic information generated in the process.

Using Diagnostics you can:

- Monitor the general flow of control as the WebLogic Mobility Server processes requests and responses.
- Troubleshoot a page of content that is not being transformed correctly.
- Monitor URL rewriting - see how URLs get rewritten.
- View requested content in its pre-transformed state.
- Analyze how WebLogic Mobility Server is performing page-splitting and dealing with author-preferences.
- Diagnose interaction between the application and the DeliveryContext API.
- Analyze the headers and cookies that the application is sending back to the browser.
- Inspect the WebLogic Mobility Server process for inserting next-page links into transformed content

Diagnostic Tools

There are two diagnostic tools available:

- The Diagnostic Console

  This provides a series of dialogs for entering the information required to generate the diagnostic messages. You can use it to simulate device requests by manually creating HTTP requests or you can use it as a proxy between the device (or device emulator) and WebLogic Mobility Server to monitor the request / response cycle.

- The Diagnostic Command-Line Interface (CLI)

  This tool requires you to pass the diagnostic parameters via the command line, or from a file.

The CLI differs from the Console in that it allows you to track the WebLogic Mobility Server activity across multiple users and requests. The Console deals with single requests only.

This section describes the steps required to run both the Diagnostic Console and the Diagnostic Command-Line Interface tools.

Ensure Diagnostics is Installed in Your Web Application

To ensure diagnostics is installed in your web application check that the diagnostics entry in the *web.xml* file is uncommented. See the section “Example web.xml: Enable Diagnostics”.

Example web.xml: Diagnostics Servlet

```
<servlet>
   <servlet-name>DiagnosticsServlet</servlet-name>
   <servletclass>com.mobileaware.diagnostics.http.server.DiagnosticsServlet
   </servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>DiagnosticsServlet</servlet-name>
   <url-pattern>/Diagnostics/*</url-pattern>
</servlet-mapping>
```

About the Diagnostics Server

When connecting to BEA WebLogic Server, the CLI uses t3 (jndi protocol) to transport diagnostic messages. This is enabled when the *web.xml* file is configured for diagnostics.

Start the Diagnostic Console

The startup file for the Diagnostic Console (*DiagnosticsConsole.exe*) is located in the **<WLMS_install_directory>\applications** directory.

- On Windows – run *DiagnosticsConsole.exe*
  Alternatively, if WebLogic Mobility Server is accessible from the Windows Start Menu (as chosen on installation), the Diagnostic Console is in the **Applications** sub-folder.
- On UNIX – run *DiagnosticsConsole*.

Start the Diagnostic Command Line

The startup script for the Diagnostic Command-Line Interface (CLI) is located in the **<WLMS_install_directory>\applications** directory. (This script is enabled by default when WebLogic Mobility Server is installed.)

Windows —*DiagnosticsTextUI.exe*

UNIX —*DiagnosticsTextUI*.

Use the Diagnostic Console

The Diagnostic Console is used to monitor WebLogic Mobility Server as it processes single HTTP request / responses. The console provides an easy-to-use interface for entering the various settings required to create the HTTP request and to view the response it generates. It can also be used to modify requests from devices and browsers.

Configure the Diagnostics Console

Before you start, you need to configure the Diagnostic Console so that it can communicate with WebLogic Mobility Server.

To Configure the Console

Launch the Diagnostic Console and choose **Settings → Settings**. When the dialog displays, make the changes to the appropriate fields.

Diagnostics Settings – Systems Tab

Setting	Description
Server	The address of the server that the Diagnostic Console is to connect with. For example, developer.myserver.com
Port	The port number of WebLogic Mobility Server. For example, 7001 or 8080.
Listen Port	The port number where the Diagnostic console should listen for new device/browser-based requests when it is configured in interception mode. The default is 4444.
Protocol	The Diagnostic Console protocol is currently limited to HTTP.
Username/Password	The username and password that the Diagnostic Console will use when connecting to diagnostics. This is required if the Diagnostic Console is configured as protected in the web application descriptor - *web.xml*. Normally they are both clear.
Deployed Path	The path to the web application that is configured for Diagnostics. The default deployed path is created by adding **/Diagnostics** to the webapp path.

Diagnostics Settings – Setting Preferences Tab

Setting	Description
Bypass Patterns	Request Patterns to be ignored by the Diagnostic Console. Typically, these patterns will be media files, such as JPG and GIF files. These patterns will not register as a request and therefore the associated response will not be displayed in the Request and Response Panels of the IDE respectively. Use a semi-colon to separate patterns. Example: *.gif*;*.jpeg*;*.wbmp*
Logging	To enable logging, select the **Logging** check box. Specify the name and location of the log files that you want to use for logging requests and responses.
Default Text Editor	The text editor to use when editing the response content. for example *notepad.exe*

These settings are saved when closing the Diagnostic Console and restored when the Console is restarted.

Note: When the settings on the Settings Preference tab are altered, the log files are cleared.

Simulate HTTP Requests

The Diagnostic Console enables you to test the effect of a HTTP request, and examine its response, without having to use a device or a device emulator.

Choose **File → Request**. This displays the Request Panel where you can create your HTTP Request.

From here you can create a HTTP request that includes:

- The protocol and version to be used: HTTP Version 1.0 and Version 1.1.
- The type of request to be constructed: GET or POST
- The destination URL.
- The device being emulated.
- Cookies, query strings and any parameters that you want to include.

Specify a URL Request

When specifying a destination URL, ensure it is relative to the web application you are accessing.

For example, if the complete URL is:

```
http://server:port/news/index.jsp
```

then type in the following into the URL Request field:

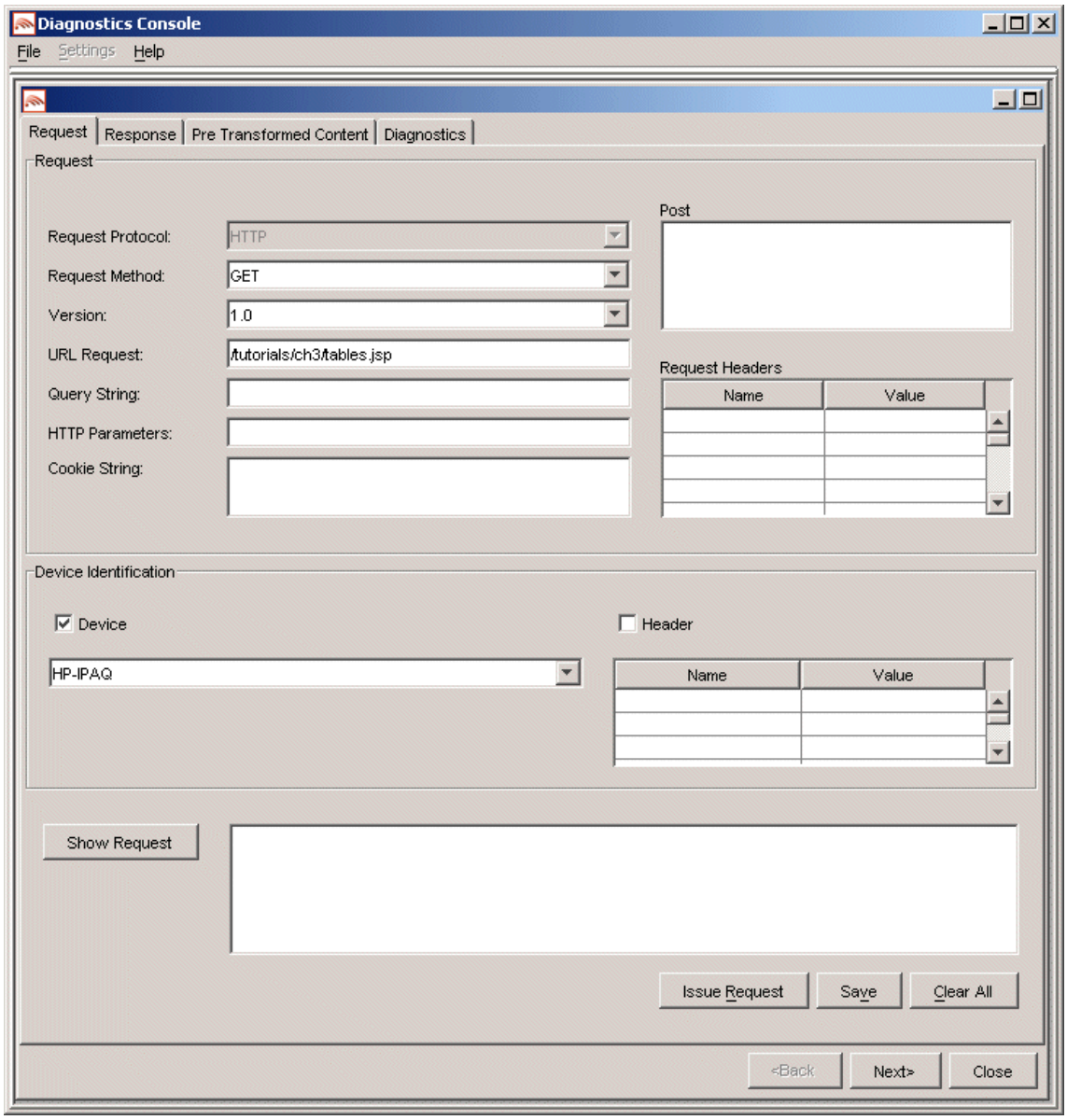
```
/news/index.jsp
```

WebLogic Mobility Server concatenates the Server and Port values provided earlier with the request field to generate the target URL:

```
http://server:port/news/index.jsp
```

The other parts of the full URL will be taken directly from the other input boxes in the panel or derived from their contents.

The Diagnostics Console



Insert a Query String

Use the Query String field to insert queries of the form: ?name=tonks. Do not use the “?” as this is inserted automatically.

Insert HTTP Parameters

If HTTP parameters are required, enter them as name-value pairs. For example

```
jsessionid=1234
```

In a HTTP Request, HTTP parameters are usually appended to the URL request and separated by a semicolon ";". For example,

```
http://server:port/news/index.jsp;jsessionid=1234?name=fred
```

Any number of parameters can be included, each separated by a semicolon.

Insert Cookies

Although you can add cookie headers directly in the Request Headers field, it's recommended you use the Cookie String field to insert cookies in the HTTP request header. The "Cookie:" header name is not required. It is added automatically to the request if cookies are used.

For example if you want to add the following to a request:

```
Cookie: Customer="Person_1"
type:
    Customer="Person_1"
in the Cookie field.
If your cookie consists of multiple parts, use a semi-colon as a delimiter.
For example to add
    Cookie:Ver="1"; Customer="Person_1"; $Path="/acme"
type the following in the cookie field:
    Ver="1"; Customer="Person_1"; $Path="/acme"
```

Insert Other Headers

Use the Request Headers field for additional HTTP request headers that take the form of a name-value pair. For example the header "Pragma: no-cache" can be represented by adding Pragma to the Name area, and no-cache to the Value area.

The maximum number of name-value pairs that can be added is 20.

Note: There is no need to enter a "Host:" header, as the WebLogic Mobility Server Application Server console generates this automatically.

Specify POST Requests

Use the POST field to enter text for POST requests. The contents of this text field will be appended to the end of a POST request. Currently, only text content is supported. This field is used when the Request Method is POST; otherwise it is ignored.

Specify the Target Device

Use the Device Header field to specify the device being targeted. This ensures WebLogic Mobility Server generates the appropriate response to any HTTP requests coming from the Device Console. You can select from a set of device types or type the header(s) in manually

Choose a Pre-Defined Device

Select the Device option to use a pre-defined device type. The Device dropdown list becomes available.

Select the device you want to target from the Device list.

Specify a Custom Device

If the device you want to target is not in the pre-defined list, you can type in the header(s) manually. Typically, you will use this option when working with experimental devices that have not yet been added to the Device Repository.

Select the Manual option and type in the header(s) manually.

Preview the HTTP Request

Click **Show Request** on the Request Panel to preview the full HTTP request that will be sent to WebLogic Mobility Server.

If you are dissatisfied with what you see, modify the settings in the Request Panel and click **Show Reques**t again.

Save the Request Pattern

You can save the request pattern as a text file.

Click **Save** and provide the filename and location when requested.

Clear the Request Pattern

Click **Clear All** to clear the fields on the panel.

Issue an HTTP Request

Press the Issue Request button to send the HTTP request to WebLogic Mobility Server. The response will appear in the Response Panel.

The **Issue Request** button is disabled when a request is sent. When the request/response cycle is completed, the button is enabled again. Click **Next** or click on the **Response** tab to go to the response view.

Use the Response View

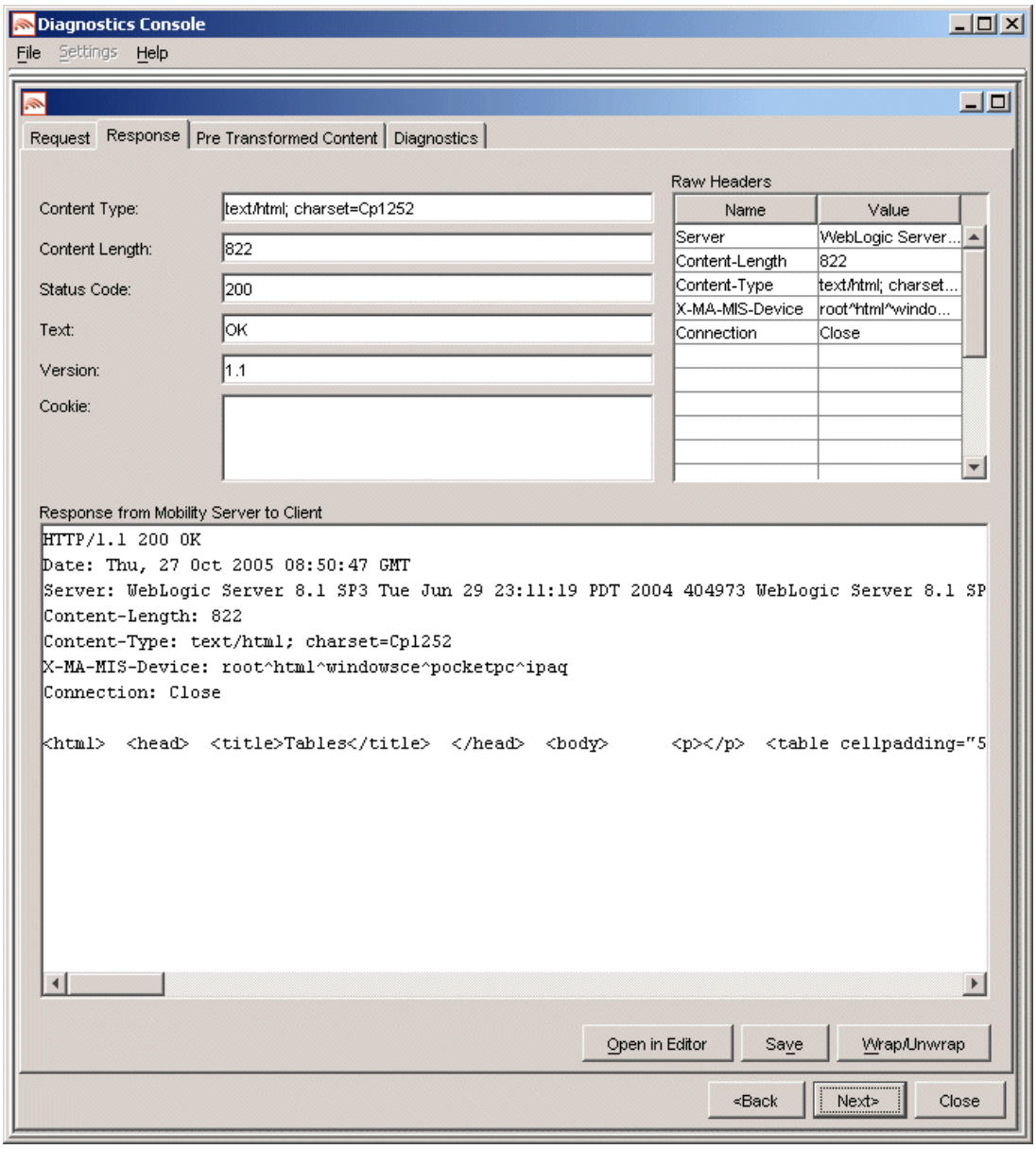
The Response Panel displays the HTTP response from WebLogic Mobility Server to the last successful request issued. This response contains the content transformed by WebLogic Mobility Server and displays a number of attributes associated with the response.

Response View

Field	Description
Content Type	Displays the contents of the “Content-Type” header derived from the response for example `text/html`. **Note**: If this header is missing in the response the text field will be clear.
Content Length	Displays the contents of the “Content-Length” header derived from the response for example `1024`. Servers do not always return a “Content-Length” header so this field will be empty if the header is missing in the response.
Status Code	Displays the status code returned. The typical return code for successful content retrieval is 200. The return code 404 flags content that could not be found.

Text	Displays the text of the response line. This is also known as the Reason Phrase. It will contain text to describe the outcome of the request. For example, OK or Not Found
Version	Displays the HTTP protocol version used in the response returned by the server.
Cookie	Displays any cookies contained in the response header.
Raw Headers	Displays a list of all the headers returned by WebLogic Mobility Server.
Response from WebLogic Mobility Server to Client	Displays the full HTTP Response (including the body) from the server.

Response content



Save the Response Content

You can save the response content as a text file.

Click **Save** and provide the filename and location when requested.

Edit the Response Content

To edit the response content:

Click **Open** in Editor button.

View the Response Content

Click **Wrap/Unwrap** to wrap the text in the window so that it is easier to view. The button acts as a toggle: select it again to reverse the wrapping action.

Failed Requests

Requests issued to WebLogic Mobility Server may fail for a number of reasons for example invalid request format, invalid URL. However if the request should fail because no response has been received from WebLogic Mobility Server, or the Diagnostic Console cannot communicate with the server, the Console detects these failures as timeouts. On timeout, if the request is unsuccessful the content returned will be appropriate to the device (or simulated device) making the request (such as WML or HTML).

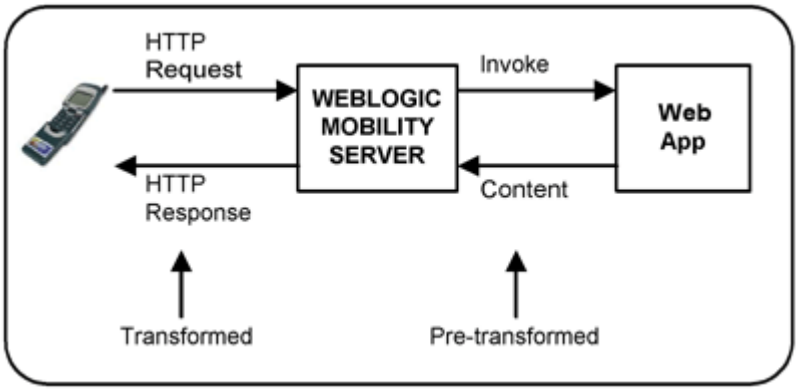
Note: If the requested server returns any HTTP response, even a HTTP 404 Response, the Diagnostic Console treats this as a valid response from the server.

Pre-Transformed Content View

Use the Pre-transformed Panel to view the marked up content before WebLogic Mobility Server transforms it. This will be useful for the user in that the content can be observed before (pre-transformed) and after (response) transformation.

Note that when a large browser device requests a WebLogic Mobility Server JSP taglib page, the content delivered to the pre-transformed content panel is the response from the application server rather than the source of the taglib page. In this scenario, most of the processing will have already been done by the taglib.

Content Transformation



Save Pre-Transformed Content

You can save the pre-transformed content as a text file.

Click **Save** and provide the filename and location when requested.

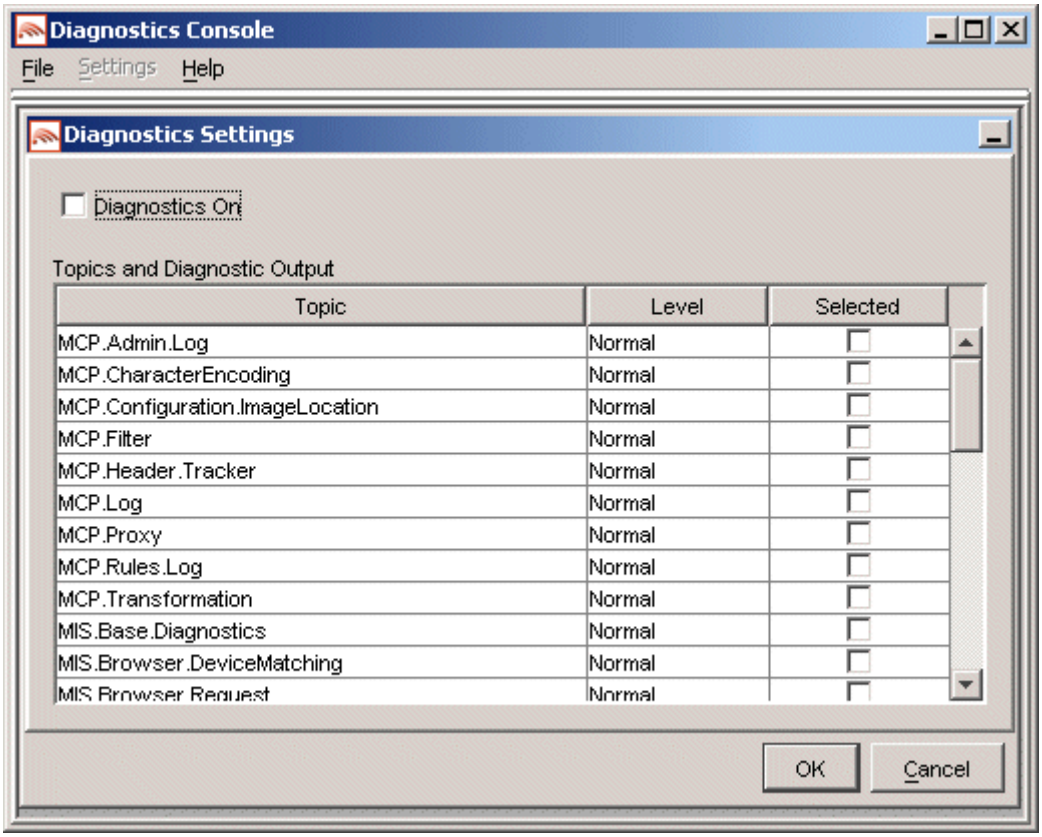
Work with Diagnostics

You can control the type and amount of diagnostic information generated from WebLogic Mobility Server. These settings are saved when you exit the application and are restored when you restart it.

1. Choose **Settings** → **Diagnostic Settings**
2. Select the **Diagnostics On** check box
3. Select the topic and level of output — Normal or Verbose — you want from the Diagnostic Output list.

Note: A standard set of diagnostic messages will be output even if no diagnostic information settings are selected.

Diagnostic Settings



View Diagnostic Information

Diagnostic messages, and any error messages produced if the request fails, are displayed in the Diagnostics panel.

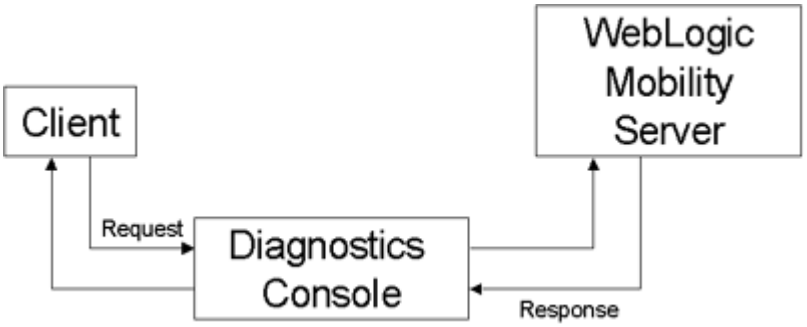
Standard Attributes Provided with Diagnostic Output

Attribute	Description
Incoming URL	This is the URL of the incoming request to WebLogic Mobility Server. For example, 199.168.5.30:7001/login.jsp
Device Pattern	This is the matched device pattern. For example root^WML^ericssonr380
Device	This is the identified device. For example, EricssonR380
Session ID	This is the session id.

Interception: Use the Diagnostics Console as a Proxy

The Diagnostic Console can be used as a proxy, so that requests can be sent from a device or emulator, passed through the Diagnostic Console and on to WebLogic Mobility Server. The response is then returned through the console to the device.

Using the Diagnostics Console as a Proxy



If bypass patterns have been specified in the Diagnostic Console settings panel, these will still pass back and forth through the proxy, but the output will not be displayed. In other words, requests for images will not generate diagnostic messages. For example, a page might contain two images and text, so you could specify that the image request / responses are not trapped. If the page had a style sheet (CSS) associated with it, you could also specify this as a bypass so that it gets passed through.

Start Interception

1. Choose **Settings → Interception** to configure and start the interception.
2. Select the device type you want to target.

   Note: For predictable results ensure you are using the appropriate browser. For example, if you specify a WAP device but are using Internet Explorer as the browser, it will not display the WAP content correctly.
3. Use your device to enter the address for the selected page. The request takes the form:
   `http://dc_host:dc_port/URL_request`

   where "dc_host" is the IP address of the machine where the Diagnostic Console is running, "dc_port" is the port on which the Diagnostic Console is listening for requests (specified in the Settings panel) for example, 4444. The "URL_request" is the name of file you are investigating relative to the web application directory.

For example,

http://199.168.5.37:4444/requested_page.jsp

When the request reaches the Diagnostic Console, the response is displayed in the Response View of the window. If the Diagnostics option is chosen, then the diagnostics messages will be displayed in Diagnostics/Pre-transformed views.

Use the Diagnostic CLI

The Diagnostic Command-Line Interface (CLI) monitors WebLogic Mobility Server as it processes multiple requests to applications. Use the CLI when you want to monitor requests being made by several users — the Diagnostic Console monitors single requests only.

The Diagnostic Command-Line Interface provides a number of parameters that enable you to track down performance or transformation issues. By default, the CLI reports all activity. You can, however, specify a filter so that only activity that matches the specified filter is reported.

As you monitor WebLogic Mobility Server, it is likely that you will create a suite of diagnostic tests, each with its own set of parameters. The CLI supports the use of parameter files — text files containing a list of parameters — so you do not have to keep retyping the parameters from the command line. You simply pass the parameter file when starting the CLI

See the end of the section for examples on how to use the CLI and to see some sample output.

Important note: The JVM on the client running the CLI must be the same version as on the server running WebLogic Mobility Server.

Use the Diagnostic CLI with WebLogic

In a WebLogic deployment the Diagnostic CLI should be installed in, and invoked from, the server environment. This tool uses Java Remote Method Invocation (RMI), which may have implications for WebLogic users wanting to run this tool outside the production environment. If a firewall infrastructure is in place, certain ports may need to be opened between the client running the tool and the server. See the Java RMI documentation at ***http://java.sun.com/products/jdk/rmi/*** for more information.

The Diagnostic CLI requires access to *weblogic.jar* via the CLASSPATH to support command line diagnostics on WebLogic. This CLASSPATH is configured via the `lax.class.path` property in `DiagnosticsTextUI.lax`.

The installed and configured CLI can be invoked either locally at the server's console or from a remote telnet client.

Note: For the initialization of WebLogic, at least one HTTP request must have been issued to WebLogic Mobility Server before invoking the Diagnostic CLI, otherwise WebLogic will generate errors relating to the absence of a JNDI tree.

Start the Diagnostics CLI

The Diagnostic CLI is located in the folder **<WLMS_install_directory>/applications**

- On WINDOWS — run *DiagnosticsTextUI.exe*
- On UNIX — run *DiagnosticsTextUI*.

Usage

The command line takes the form:

```
DiagnosticsTextUI -appserver AppServer [options]
(to listen to diagnostics)
```

OR

```
DiagnosticsTextUI -usage [option]
(to view usage)
```

Diagnostics Usage

To:	**Specify this parameter:**
Establish a connection to the diagnostic engine	- username - password - jndiprotocol - host - port - appserver - url - deploypath/contextpath
Use parameter files to pass parameters to the command line	- paraminfile - paramoutfile
Choose which diagnostics to monitor	- topicname
Get help on using the CLI options	- usage

Get Help

To display the startup options available, start the CLI as follows:

```
DiagnosticsTextUI –usage
```

To get help on a specific option, type:

```
DiagnosticsTextUI -usage [option]
```

Example:

```
DiagnosticsTextUI -usage paraminfile
```

Specify the Connection

You need to specify a number of connection parameters in order to connect WebLogic Mobility Server with the diagnostic messaging:

Connection Parameters

Option	Description
- username	The username to connect to WebLogic Mobility Server
- password	The Password to connect to WebLogic Mobility Server
- host	Host name or IP address of the application server
- jndiprotocol	For WebLogic, specify "-jndiprotocol t3"
- port	Port of the application server
- url	For WebLogic only, a URL can optionally be supplied *instead* of the host/port/ jndiprotocol combination of parameters. An example of this is: "-url t3://server:port"
- appserver	Type of application server being connected to: WebLogic.
- deploypath	This field is not required for BEA WebLogic.
- contextpath	Path to where the web application is deployed with **Diagnostics** appended. For example, **news/Diagnostics** where **news** is the path to the deployed web application (only required for WebLogic connections)

Use Parameter Files

Rather than typing in a long list of parameters, you can save time and effort by storing your parameters in a file and passing this file to the Diagnostic CLI. For an example, see "Example 2: Use a Parameter File" later in this section.

Parameter files are plain text files, with each parameter on a separate line. The file can contain all parameters except the username and password.

There are two ways to use parameter files:

- –paraminfile file: Specify a file containing the parameters to be read in
- –paramoutfile file: Specify a file to capture the parameters you enter on the command line. This file can be subsequently used to read in the parameters.

Filter Content

By default, CLI monitors and reports on all activity. However, you can use the -filter parameter to listen for and report on specific activity, thereby screening everything else out. For example, you can choose to listen for activity relating to a specific URL

Filtering Content

To display messages:	**Specify this FILTER option:**
For a specific request header	requestHeader:<headerName>
For a specific unique request header that matches selected value	uniqueRequestHeader:<headerName>
Containing a specific request parameter	requestParameter:<paramName>=<value>
For a specific unique request parameter that matches the selected value	uniqueRequestParameter:<paramName>
For a particular URL	url:<value>

Specify Diagnostic Messages

Specify, at the end of the command line, the diagnostic messages (topics) you want to listen to. Additionally, you can specify whether you want normal or verbose output. A full list of topics is described in the section.

Example: MIS.Client verbose

Example Usage

The following examples show how the CLI can be used.

Example 1: Generic Usage

WebLogic Example

```
DiagnosticsTextUI –appserver weblogic –username AdminUser
–password AdminPass –host
Server1 –port 7001 –jndiprotocol t3 -contextpath /news/Diagnostics
MIS.General.FlowOfControl
```

Example 2: Use a Parameter File

The following example illustrates how a parameter file is used to specify the connection settings and instruct the diagnostics to monitor the flow of control with verbose messaging turned on.

```
DiagnosticsTextUI –appserver weblogic –password AdminPass
–paraminfile param.txt –contextpath /news/Diagnostics
```

where param.txt contains:

```
-username adminName
-host Server1
-port 7001
-jndiprotocol t3
MIS.General.FlowOfControl verbose
```

Example 3: Get Help on the Filter Parameter

The following example displays the options available for the filter parameter:

```
DiagnosticsTextUI –usage filter
```

Diagnose Problems

This section describes some of the scenarios you may encounter when working with WebLogic Mobility Server and how to approach them.

Identify the Source of a Problem

When you encounter a problem with a request first confirm that WebLogic Mobility Server is running. If it is, see the next section in this section “Identify Connection Problems”.

If WebLogic Mobility Server is running, but you are getting transformation or content errors, see “Resolve WebLogic Mobility Server Error Pages”.

If the error is common across all devices, see “Resolve WebLogic Mobility Server Error Pages”.

Identify Connection Problems

If you receive a message stating that “a connection could not be established” then:

- Verify that you are using correct values for the host name and port in the request.
- Check that your web application server is running.
- Check that associated components, such as the BEA WebLogic Portal, are online.
- Check that WebLogic Mobility Server is running.

If you are satisfied that all applications and components are running correctly, but there are still problems, check the database connection settings in the *mis.properties* file located in your webapps **WEB-INF/classes** folder.

Resolve WebLogic Mobility Server Error Pages

WebLogic Mobility Server generates an error page when it is unable to retrieve content or encounters problems when it is transforming content.

Problems Retrieving Content

WebLogic Mobility Server generates an error page if it is unable to retrieve the content. This occurs in the following circumstances:

- The requested page does not exist.
- The URL has been misspelled.

Resolve Badly Formed Content / Faulty JSP Code

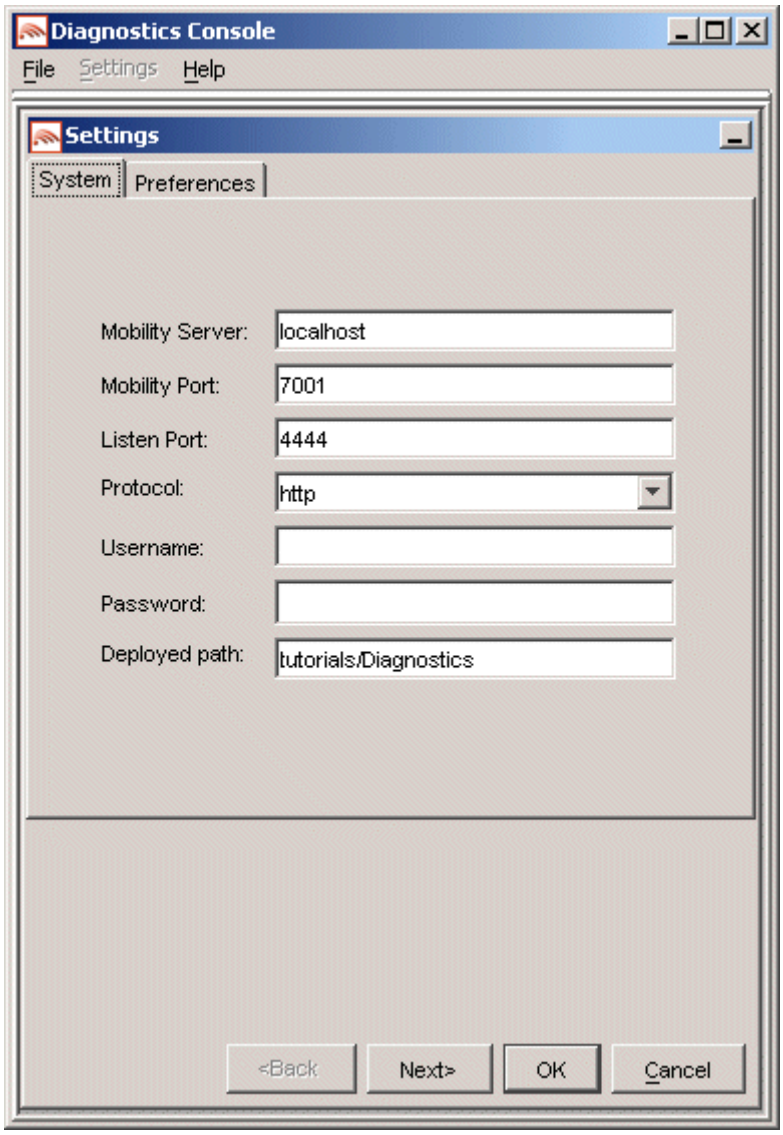
WebLogic Mobility Server generates an error page when it receives a page that contains poorly formed content. Typically, the error message will indicate missing elements or attributes, such as a missing “/” to close a tag properly. There are two scenarios where this is likely to occur:

1. When you first mark up an otherwise functional JSP file for mobilization. This is likely because of the more rigorous validity checking done for XHTML. Running an external validator will usually catch mark-up errors like these.
2. When you have used JSP scripting elements to dynamically generate XHTML output. The validation tool will not catch errors in these elements as they only get output at runtime.

If validating the JSP source does not resolve the problem:

1. Switch on Interception in the Diagnostic Console and set the listen port on the Settings window in the Diagnostic Console. Ensure that the value set for the listen port is used in your HTTP request. For example, if the listen port is set to 4444, enter the request as "`http://test:4444/news/index.jsp`".
3. Make a request for the page and save the pre-transformed content.
4. Open the file you just saved, remove the header text and run an external validator on it. This may highlight XHTML errors being dynamically generated.

The Diagnostic Console Properties Window



Resolve Database Errors

WebLogic Mobility Server generates an error page when it has problems connecting to the Device Repository. It requires this database in order to recognize the requesting device and decide how to transform the content to match the device's hardware and software capabilities.

Connection-Related Messages

- Confirm the database is actually running.
- Check the database settings (url, driver, user and password) in the *mis.properties* file.

SQL-Related Messages

SQL-related error and exception messages are more likely to denote queries that contain incorrect table, column or row names.

Device-Specific Errors

Confirm whether the problem manifests itself on all devices or only on a particular device. If the problem occurs on all devices, it is probably a content mark-up problem. If the problem occurs with a specific device, then your content may be stretching the limitations of the device or encountering conflicts with the mark-up that the device supports.

Monitor Diagnostic Output

This section describes:

- diagnostic output available and how it is organized into categories known as topics
- information generated for each topic
- appropriate action to take in response to the diagnostic output

About Diagnostic Topics

The diagnostic output can be broken down into a series of “topics”, where each topic relates to a different area within WebLogic Mobility Server, such as session management, cookie handling, device management or the transformation engine.

Each topic has an associated set of messages that indicate the actions being taken and the conditions encountered as the requests and responses progress through WebLogic Mobility Server. Some messages may be associated with more than one topic.

With approximately 400 individual diagnostic messages available, the diagnostic output can be quite substantial. To restrict the amount of messages generated, you can choose to generate “normal” or “verbose” output for each topic. Messages that are associated with more than one topic may be treated as part of “normal” output by one topic and “verbose” output by another.

Example: Normal vs. Verbose Output

	Message 1		**Message 2**		**Message n**	
	Normal	Verbose	Normal	Verbose	Normal	Verbose
Topic 1	X			X		
Topic 2		X				
Topic 3					X	

Diagnostic Topic Categories

Diagnostic information is organized into the following broad categories; each of which has a number of topics that can be individually selected:

- **Client Transactions** — Monitors transactions relating to cookies, headers, parameters and sessions
- **Database Transactions** — Monitors database transactions with devices, and with the system, including connections
- **Devices** — Monitors device transactions, such as recognition
- **Web Applications** — Monitors interaction with the web application (for example JSP page), such as headers, cookies and pre-transformed content
- **Transformation** — Monitors processing associated with content transformation, such as the application of Mobility Controls, Pagination, URL rewriting, Table Transformation and Layout selection

- **DeliveryContext API** — Monitors processing related to the use of DeliveryContext API methods
- **Flow Of Control** — Monitors the general progress of WebLogic Mobility Server as it progresses through the request / response cycle

Monitor Client Transactions

Selecting Client Transaction topics will generate messages relating to headers, parameters, sessions, and cookies.

Select these topics to monitor client transactions:

- MIS.Client
- MIS.Headers
- MIS.Browser.SessionLifeCycle

Interpret URL Rewriting

There are two common ways of passing session keys to the client and back:

- Cookies
- URL rewriting.

Cookies are used by default; URL rewriting occurs if the browser does not support cookies. During URL rewriting, all WebLogic Mobility Server-generated URLs that are included in the server response are encoded to contain the session key.

The diagnostic message for URL rewriting indicates the URL before and after the rewriting. It takes the form:

```
URL Before: url_before. URL After: url_after
```

Interpret Cookie-Related Messages

Cookie-related messages will indicate whether the request contains cookies or not. If the request contains cookies, the messages will indicate the incoming cookies. Each message will indicate the cookie and its corresponding value in the form: cookie: value.

Interpret Session-Related Messages

Session related messages will indicate when the session was created and the ID it was assigned.

Monitor Database Transactions

When you monitor database transactions you can track the interaction with:

- information relating to devices and web applications
- the connection pool
- SQL Queries

Select these topics to monitor database transactions:

- MIS.Database.Query
- MIS.Database.Cache
- MIS.Database.Device
- MIS.Database.ConnectionPool

Interpret Connection Pool Messages

Diagnostic messages allow the state of the connection pool to be viewed at any time, with every connection having a description that identifies its use. Diagnostic messages will provide information about:

- creating, extending and removing connections from a pool
- waiting on, getting, and returning connections
- problems encountered making connections

Monitor Devices

To monitor device-related activity, select the following topic:

- MIS.Device

Interpret Device Repository-related Messages

Device messages relating to the database will indicate that a device is being looked up in the database, whether it exists (“found”) or not, and any attributes that are being retrieved.

Monitor Web Application Transactions

When monitoring web application transactions, you can select a topic to track transaction activity.

Additionally, you can track the interaction with cookies and headers: for a description of messages associated with these topics, see “Monitor Client Transactions” on page 152.

Select this topic to monitor web applications:

- MIS.Service

Monitor Transformation

Transformation messages will indicate which Device Transformation Map is being used, whether a document is paginated or not (“sub-pages”) and any difficulties encountered when resolving internal links to any sub-pages created.

Select these topics to monitor transformation:

- MIS.Transformation
- MIS.Transformation.DTM
- MIS.Transformation.Pagination
- MIS.Transformation.URLRewriting
- MIS.Transformation.WML.Tables
- MIS.Navigation
- MIS.FormPagination

Interpret Pagination-Related Messages

Since WML devices have a limited capacity to store and display content, WebLogic Mobility Server breaks the content into sub-pages, creates the appropriate links between the pages to ensure a seamless delivery, and caches the sub-pages so that they are ready for delivery when the WML device makes a subsequent request.

Diagnostic messages will indicate the number of sub-pages created, and report on the success or failure in locating a sub-page for delivery.

Interpret Table-Related Messages

If your content contains tables, WebLogic Mobility Server will generate messages to indicate whether a table is contained within the `<mm-table-model>` tag, if the table content is poorly formed, if there is a lack of uniformity in the row / column arrangement (for example, a different number of `<td>` elements in each row). It will also indicate if there is too much content for it to handle in a table row.

Monitor Browser Activity

Monitors activity with the browser.

Select these topics to monitor Browser Activity:

- MIS.Base.Diagnostics
- MIS.Browser.DeviceMatching

- MIS.Browser.Request
- MIS.Browser.SessionLifeCycle

Monitor Flow of Control

Monitoring general activity turns on messages that indicate the flow of control, including the start-up sequence, as WebLogic Mobility Server processes requests and responses, requests received and finished events. It also provides cookies and header related information.

Select these topics to monitor general activity:

- MIS.General.Cookies
- MIS.General.FlowOfControl
- MIS.General.Headers
- MIS.General.Startup

Monitor the Mobility Filter

The mobility filter topic describes aspects of the processed request. It provides details of the following: entry and exit of the request, device recognition, database type being used, request parameters and query string, and the headers that are being passed, modified or passed through.

Select this topic to monitor the mobility filter:

- MIS.MobilityFilter

Monitor Response Received

The following two topics detail aspects of the response received from a request:

- MIS.Response.Received.Content

  This topic details the content received back from a request. This is the content, as it exists, before WebLogic Mobility Server begins the transformation process.

- MIS.Response.Received.ErrorResponse

  This topic details the received error response in the event of there being a problem processing a request from WebLogic Mobility Server, that is, if the HTTP Request Status code indicates a problem.

Monitor the JSP Tag Library

Monitoring the JSP Tag Library turns on messages that indicate when a tag library tag is encountered, what output it generates, and when it is being released from memory. In verbose mode, it is possible to see the results of individual `where` attribute and CDATA process as well.

Select this topic to monitor the JSP Tag Library:

- MIS.JSPTagLibrary

Configure WebLogic Mobility Server to send logging messages to Log4J

WebLogic Mobility Server also supports monitoring of diagnostic output via an external logging mechanism such as Log4J. A GenericLogMonitor facility has been provided to enable output of diagnostic messages to a tool such as Log4J. For or details on using Log4J to monitor WebLogic Mobility Server Diagnostic output, see “Appendix D - Use the Generic Log Monitor Facility with Log4J”.

Exception Handling

There are two separate operational modes in WebLogic Mobility Server. These are as follows:

- “Development” mode
- “Production” mode

To indicate which mode WebLogic Mobility Server is to operate in, configure the `operation.mode` setting in the *mis.properties* file. Setting the mode to “development” provides detailed informative warning messages to enable content developers to tune and troubleshoot content during the development phase.

Example

```
operation.mode: development
```

By default, the WebLogic Mobility Server operation mode is set to “production”. In production mode, WebLogic Mobility Server provides error messages only. Warnings are not provided.

Note: To change from one mode of operation to the other, edit the *mis.properties* file accordingly and then re-start WebLogic Mobility Server.

Development Mode

In development mode, WebLogic Mobility Server provides warnings and error messages to the web application developer. These warnings and error messages indicate instances of incorrect content or incorrect development practice.

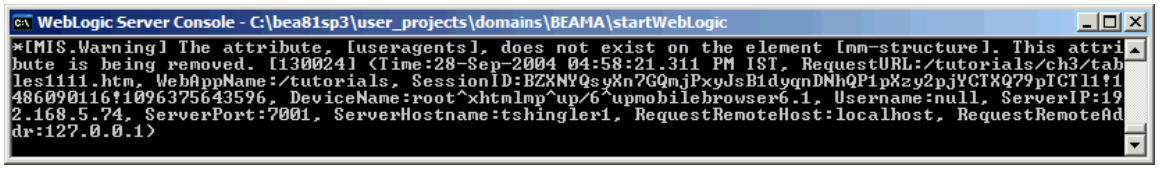
Hello World Example

The following “Hello World” example illustrates that when WebLogic Mobility Server processes this code in development mode, a warning is displayed.

Note: The console screen message states that WebLogic Mobility Server does not support the useragents attribute.

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//MOBILEAWARE//DTD MMXHTML 1.2//EN"
"http://www.mobileaware.com/DTD/mmxhtml_1.2.dtd">
<html>
<head>
<title>Project 2.1 - Hello World</title></head>
<mm-structure useragents="smallbrowser/wml" id="structure_1"
where="IsMenuDriven">
    <mm-group-ref idref="gp_101" type="normal" depth="0" display="all"/>
</mm-structure>
<body bgcolor="#FFFFFF" text="#000000">
    <i>This file is best viewed with a WML browser</i><br/>
    <mm-group id="gp_101" title="Hello World">
    <mm-head id="hd_101" useradded="no">Welcome</mm-head>
    <mm-body id="bd_101" idref="hd_101">
        <p>Hello World! </p>
        <mm-exclude where="IsFullBrowser">
            <p>Welcome to your first mobile demo.</p>
        </mm-exclude>
    </mm-body>
    </mm-group>
</body>
</html>
```

Console output in Development Mode



Production Mode

In production mode WebLogic Mobility Server provides error messages only. Warnings are not provided.

Hello World Example

The “Hello World” example illustrates that in production mode, no errors are produced in the console. The output is shown here:

Hello World result on WML emulator

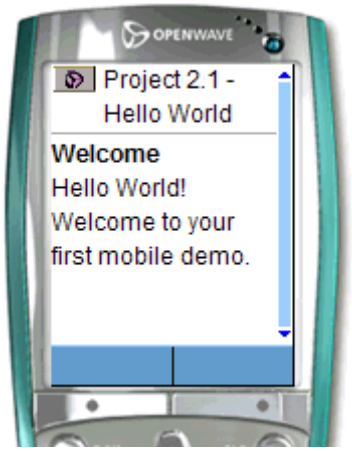


Image Courtesy of Openwave Systems Inc

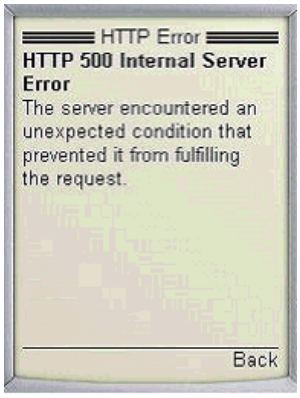
Example

The “Hello World” example includes an invalid `where` condition which specifies PDAs only. Because it doesn’t include a `where` condition for menu-driven devices, an error will occur when a WAP browser requests the page.

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//MOBILEAWARE//DTD MMXHTML 1.2//EN"
"http://www.mobileaware.com/DTD/mmxhtml_1.2.dtd">
<html>
<head>
<title>Project 2.1 - Hello World</title>
</head>
<mm-structure where="IsPDA" id="structure_1">
    <mm-group-ref idref="gp_101" type="normal" depth="0" display="all"/>
</mm-structure>
<body bgcolor="#FFFFFF" text="#000000">
    <i>This file is best viewed with a WML browser</i><br/>
    <mm-group id="gp_101" title="Hello World">
    <mm-head id="hd_101" useradded="no">Welcome</mm-head>
    <mm-body id="bd_101" idref="hd_101">
        <p>Hello World! </p>
        <mm-exclude where="IsFullBrowser">
            <p>Welcome to your first mobile demo.</p>
        </mm-exclude>
    </mm-body>
</mm-group>
</body>
</html>
```

Console Output in Development Mode

Emulator Output in Development Mode



Part VII Glossary

Glossary

Term	Description
Administrator	The Administrator is a general term for someone who logs into the Administration Console and uses it to create, modify, and remove devices.
CSS (Cascading Style Sheet)	CSS is a feature of HTML that gives both Web site developers and users more control over how pages are displayed. With CSS, designers and users can create style sheets that define how different elements, such as headers and links, appear. These style sheets can then be applied to any Web page. The term *cascading* derives from the fact that multiple style sheets can be applied to the same Web page. CSS was developed by the Worldwide Web Consortium (W3C). The specification is still evolving and is not fully supported by any current Web browsers.
Delivery Context API	The delivery context API is a set of methods used for accessing device information from the Device Repository. There are two methods used to access this information. For CC/PP attributes (which have a name beginning with UAProf), the JSR188 API can be used. For accessing both the Mobility proprietary attributes and the CC/PP attributes, the WebLogic Mobility Server Delivery Context API can be used. This is a development package that provides a list of public access JSP methods to the Device Repository. This database maintains the profiles on the devices that are recognized by WebLogic Mobility Server.
Device	A device is the end-component that receives transformed web application content from WebLogic Mobility Server. Devices include WAP phones (for example Nokia 7110), PDAs (iPAQ, HP Jornada), and PCs. A device can be either a specific device or a class of devices. If a device can contain other members, it is a class, otherwise it is a specific device.
Device Repository	The Device Repository is used to store device browser characteristics. A comprehensive set of device attributes is created when the product is installed. These characteristics are entered in the Administration Console so that the WebLogic Mobility Server content handling system can deal with the device interaction in a manner suited to the end-user's device. In this way, WebLogic Mobility Server uses device characteristics to maximize the content presentation to the end-user’s device. The Device Repository is compliant with CC/PP standards. It consists of both UAProf attributes and Mobility proprietary attributes.
Diagnostics Console	The Diagnostics Console provides an intuitive graphical

	mechanism for content developers to obtain information about what WebLogic Mobility Server is doing during the processing of a request and the transformation of content. This includes the ability to detect pre-transformed content, post-transformed content and URL header/body information.
DNS	The Domain Name System (DNS) is a distributed Internet directory service. DNS is used mostly to translate between domain names and IP addresses, and to control Internet e-mail delivery. Most Internet services rely on DNS to work, and if DNS fails, web sites cannot be located and e-mail delivery stalls.
DTD	Document Type Definition. A file that contains the rules for valid syntax, format, and structure for defining the mark-up elements in an XML document. Standard vocabularies are summarized in DTDs, formal grammars that declare tags and their structural relations. DTDs are available for many domains, including electronic commerce (for example, OTP, XML-EDI), science (for example, MathML, Chemical ML), synchronized multimedia (for example, SMIL), software documentation (for example, DocBook), or agent technology (for example, WIDL). See also mmXHTML DTD.
End-User	A person accessing web information. For example, a field engineer traveling to various customer sites accessing site-based information and applications via PDA (Personal Digital Assistant).
mis.properties	This is a plain text file located in the **WEB-INF/classes** folder of the webapps running WebLogic Mobility Server. It contains the various properties that are used to manage the behavior of WebLogic Mobility Server. The file can be edited in any text editor. Some of these properties will have been set during the install process, while others can be configured later to further tailor the behavior of WebLogic Mobility Server.
HTML	HyperText Mark-up Language. The authoring language used to create documents on the World Wide Web. HTML is similar to SGML, although it is not a strict subset. HTML defines the structure and layout of a Web document by using a variety of tags and attributes. The correct structure for an HTML document starts with <html><head>(enter here what document is about)</head><body> and ends with </body></html>. All the information you'd like to include in your Web page fits in between the <body> and </body> tags. There are hundreds of other tags used to format and layout the information in a Web page. For instance, <p> is used to make paragraphs and <I> … </I>is used to italicize fonts. Tags are also used to specify hypertext links. These allow Web developers to direct users to other Web pages with only a click of the mouse on either an image or word(s).
ISDN	ISDN (Integrated Services Digital Network) is an integrated digital network capable of complete digitalizing and handling of information from differing services including telephone, faxes,

	data, images, and so on
JSP (Java Server Pages)	Technology from Sun Microsystems allows Web developers and designers to rapidly develop and easily maintain, information-rich, dynamic web pages that leverage existing business systems. As part of the Java™ family, JSP technology enables rapid development of web-based applications that are platform independent. JavaServer Pages technology separates the user interface from content generation enabling designers to change the overall page layout without altering the underlying dynamic content.
MIME (Multi-purpose Internet Mail Extensions	MIME is a specification for formatting non-ASCII content so that it can be sent over the Internet. Many e-mail clients now support MIME, which enables them to send and receive graphics, audio, and video files via the Internet mail system. In addition, MIME supports messages in character sets other than ASCII. There are many predefined MIME types, such as GIF graphics files and PostScript files. It is also possible to define your own MIME types. In addition to e-mail applications, Web browsers also support various MIME types. This enables the browser to display or output files that are not in HTML format. The Internet Engineering Task Force (IETF) created the MIME standard in 1992. A new version, called S/MIME, supports encrypted messages.
mmXHTML	Multi-Mode XHTML ships with WebLogic Mobility Server. It allows content pages to be easily marked up for delivery to multiple devices. mmXHTML is a superset of XHTML. It adds a number of tags to XHTML to mark-up the structure of XHTML documents enabling a non-device-specific description of content.
mmXHTML DTD	The mmXHTML DTD (Document Type Definition) is a file that defines the mmXHTML extensions to XHTML. The mmXHTML DTD is used by WebLogic Mobility Server to validate mmXHTML documents.
WebLogic Mobility Server	WebLogic Mobility Server is a modular, extensible, carrier-grade platform for creating and delivering multi-channel data, content and applications. The platform architecture is based on Java/XML open standards and offers J2EE integration.
WebLogic Mobility Server Multi-Mode JSP Tag Library	Used for the mobilization of JSP documents. The functionality of the mmXHTML tags is replicated within a JSP tag library structure optimizing the performance of WebLogic Mobility Server when delivering JSP content to the PC channel. The WebLogic Mobility Server JSP tags mimic their mmXHTML counterparts with the exception that they use mm: instead of mm- to begin the tag name. In order for the tags to be processed, the JSP document must start with a taglib declaration and the page must be wrapped in <mm:page> tags.
SOAP	SOAP (Simple Object Access Protocol) is a lightweight protocol

	for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols.
Third Generation	3G is an ITU (International Telecommunication Union) specification for the third generation (analogue cellular was the first generation, digital PCS the second) of mobile communications technology. 3G promises increased bandwidth, up to 384 Kbps when a device is stationary or moving at pedestrian speed, 128 Kbps in a car, and 2 Mbps in fixed applications. 3G will work over wireless air interfaces such as GSM, TDMA, and CDMA. The new EDGE (Enhanced Data rates for Global Evolution) air interface has been developed specifically to meet the bandwidth needs of 3G.
UMTS (Universal Mobile Telecommunication Service)	UMTS is a Third Generation (3G) mobile technology that will deliver broadband information at speeds up to 2Mbits/sec. Besides voice and data, UMTS will deliver audio and video to wireless devices anywhere in the world through fixed, wireless and satellite systems.
URL (Uniform Resource Locator).	The content of the browser address field is a Uniform Resource Locator (URL).
URL Compression	WebLogic Mobility Server supports URL compression, which reduces the length of URLs to a minimum, thereby allowing much more WML content to be delivered to a device. This is especially relevant where the device has limited memory but could also be important where limited bandwidth is an issue. URL compression works by breaking the URL into fragments (query parameters) and replacing the fragments in the URL with shortened tokens. These shortened tokens are used by WebLogic Mobility Server to map a request generated from the replacement URL back to the original URL.
VoiceXML (Voice Extensible Mark-up Language)	VoiceXML is a standard language for building interfaces between voice-recognition software and Web content. Just as Hypertext Mark-up Language (HTML) defines the display and delivery of text and images on the Internet, VoiceXML can translate any XML-tagged Web content into a format that speech-recognition software can deliver by phone.
VoxML (Voice Mark-up Language)	The Motorola mark-up language that enables voice interaction with applications.
WAP (Wireless Application Protocol)	The de-facto world standard for wireless information and telephony services on digital mobile phones and other wireless terminals. WAP empowers mobile users of wireless devices to

	easily access live interactive information services and applications from the screens of mobile phones.
Web Deployment Descriptor (web.xml)	Before deploying a web application, you need to place a deployment descriptor (*web.xml*) into the **WEB-INF**'s directory. This file pulls together all the components of the web application to be deployed.
WBMP (Wireless Bitmap)	The WBMP format enables graphical information to be sent to a variety of handsets. It is terminal independent and describes only graphical information.
WML	WML is a mark-up language based on XML and developed specifically for wireless applications.
XHTML (Extensible Hypertext Mark-up Language)	XHTML is the first step toward a modular and extensible web based on XML (Extensible Mark-up Language). It is the reformulation of HTML 4 as an application of XML. XHTML mark-up must conform to the strict mark-up standards defined in a DTD.
XHTML MP (XHTML Mobile Profile)	XHTML MP 1.0 is the official mark-up language of WAP 2.0. It is a subset of XHTML extending XHTML Basic giving added functionality for authors developing content for a range of mobile device types supporting WAP2.0.
XML (Extensible Mark-up Language)	A web standard, similar to HTML in structure, which provides a strict set of rules for describing the meaning of data. XML is a data format for structured information interchange. Standardized by the World Wide Web Consortium (W3C), XML is the industry consensus of opinion for next generation web architecture. XML uses the concept of generic mark-up: tags that are inserted into a document, structuring it into nested elements. HTML is the most popular format using this technique. While HTML has a fixed set of tags, XML allows authors to freely choose vocabulary from their field of application to name tags.
XSL (Extensible Stylesheet Language)	XSL defines a formatting model for XML documents. It consists of two parts: a language for transforming XML documents, and an XML vocabulary for specifying formatting semantics. An XSL style sheet specifies the presentation of a class of XML documents by describing how an instance of the class is transformed into an XML document that uses the formatting vocabulary. It is left to the specific output device, to decide how the formatted result is best translated into pixels, speech, or vectors.

Part VIII Appendixes

Appendix A – Mobility Tag Reference

meta, mm:meta

Description

This tag is placed inside the document's `<head>` element. It has two WebLogic Mobility Server-specific uses:

It can be used to create a shortcut link on the Options menu of WML devices that support such a menu. This can provide a handy way for users to navigate around pages of a site.

It can be used with `<mm:cdata>` (or `<![CDATA[...]>` in `mmXHTML`) to set page-wide “wrap” or “unwrap” directives.

This is an empty tag, so it requires a slash (/) before the final angle bracket (>).

Usage A

To create a link on an Options menu

Attributes

- **name**="..."

  Assigns a name to the meta information. This will be used as the text on the menu item.

- **content**="..."

  The value that should be used for generating the link.

- **scheme=** "mmsection"

  The value of this attribute is an indicator to WebLogic Mobility Server. For WebLogic Mobility Server meta tags the value should always be "mmsection"

Example

```
<!—This will create a link with the title "Go Home" on the Options menu of some
phones.. -->
<meta name="Go Home" content="/some/url.htm" scheme="mmsection"/>
```

Usage B: To give page-wide settings to CDATA elements

Attributes

- **name**="MIS-CDATA-Control"

  This attribute must always have this value when using the meta element with `<mm:cdata>` or `<![CDATA[...]]>`. (Required)

- **content**="Unwrap | NoUnwrap"

  If set to “Unwrap”, WebLogic Mobility Server will not parse the content that is wrapped by any CDATA section on the page (unless the CDATA tag itself has contradicting instructions), but will remove the CDATA tags when delivering the page so that the wrapped content will be interpreted by the client browser. If set to “NoUnwrap”, which is the default behavior, neither WebLogic Mobility Server nor the browser will process or interpret the contents of the CDATA block.

Example using the JSP TAGLIB

```
<%-- The meta tag settings cause the first CDATA block to be unwrapped, but the
second one will not be unwrapped because the unwrap="false" in the CDATA
element overrides the meta tag instruction.--%>

<head>
<mm:meta name="MIS-CDATA-Control" content="Unwrap"/>
</head>
<body>

<mm:cdata>This content will not be parsed by the WebLogic Mobility Server
parser, but will be unwrapped and delivered to the requesting browser for
processing. Because the CDATA element has no unwrap attribute set, it looks for
the value of the content attribute of the meta element if there is
one.</mm:cdata>
 <mm:cdata unwrap="false">This content will not be parsed by the WebLogic
Mobility Server parser, and will not be processed by the requesting browser
either. The CDATA's unwrap attribute overrides the content attribute of the
meta tag.</mm:cdata>
</body>
```

Example using mmXHTML

```
<%-- This demonstrates the mmXHTML equivalent of the example above.--%>

<head>
   <meta name="MIS-CDATA-Control" content="Unwrap"/>
</head>
<body>
...
<![CDATA[This content will not be parsed by the WebLogic Mobility Server
parser, but will be unwrapped and delivered to the requesting browser for
processing. Because the CDATA element has no unwrap attribute set, it looks for
the value of the content attribute of the meta element.]]>

<![CDATA[#NOUNWRAP# This content will not be parsed by the WebLogic Mobility
Server parser, and will not be processed by the requesting browser either. The
CDATA's unwrap attribute overrides the content attribute of the meta tag.]]>

</body>
```

See Also

`mm:cdata`

mm-body, mm:body

Description

Acts as a container for the content within an <mm-group> element.

Attributes

- **id**="…"

  Assigns a unique ID to an instance of this element.

- **idref**="…"

  A reference pointing to the unique ID of the parent element. An mm-group element can have multiple mm-body elements. Each must be linked to another body element, the <mm-head>, or the <mm-group> so that they form a chain, the order of which will be the order they are displayed on a menu-driven device.

Comments

The `<mm-group>` element is used to organize your content into logical sections.

Example

```
<!— This example illustrates multiple (2) mm-body elements -->
<mm-group id="introduction" title="Introduction">
   <mm-head id="hd_introduction">Introduction </mm-head>
   <!-- Note the body is linked to the heading -->
   <mm-body id="bd_1" idref="hd_introduction">
      <p>This is paragraph 1</p>
   </mm-body>
   <!-- Note the next paragraph will only be included on full browser devices -
->
   <p>This is paragraph 2</p>
   <!–- Note the body is linked to the previous body -->
   <mm-body id="bd_2" idref="bd_1">
       <p>This is paragraph 3</p>
   </mm-body>
</mm-group>
```

See also

`mm-group, mm-head`

<![CDATA[...]]> / mm:cdata

Description

To prevent WebLogic Mobility Server from analyzing/transforming particular content, wrap this content in <![CDATA[…]]> elements. By placing the meta tag <meta…/> in the head section of the content, the enclosing CDATA tags will be removed when the content is delivered to the device. For JSP, use <mm:meta…/> and <mm:cdata> as illustrated in "<mm:cdata> USAGE."

Comments

This functionality works in conjunction with PDAs and Full Browser devices—content wrapped in <![CDATA[…]]> elements will not be delivered to MenuDriven devices.

<![CDATA[…]]> Usage

There may be circumstances when you might wish to deliver a portion of web page content that has not been subject to the WebLogic Mobility Server transformation. In mmXHTML files, the <![CDATA[…]]> tag can be used in conjunction with the <meta…/> tag to protect such content from being analyzed or transformed by WebLogic Mobility Server, and yet still have this content delivered to the device.

By extending the capabilities of the <![CDATA[…]]> tag to include unwrapping functionality, WebLogic Mobility Server will ignore the content during processing, but remove the enclosing <![CDATA[…]]> tag when delivering it to the device.

To cause your <![CDATA[…]]> tag to be unwrapped, specify it as follows:

```
<![CDATA[#UNWRAP#…]]>
```

If you have many <![CDATA[…]]> tags in a page which you wish to have unwrapped, then you can specify a <meta…/> tag as follows:

```
<meta name="MIS-CDATA-Control" content="Unwrap"/>
```

which will cause each <![CDATA[…]]> tag on the page to be treated as if #UNWRAP# were present. You may override this behavior for individual <![CDATA[…]]> tags by specifying that they are not unwrapped, as follows:

```
<![CDATA[#NOUNWRAP#…]]>
```

<mm:cdata> Usage

<mm:cdata> and <mm:meta> work together in the same way as <![CDATA[...]]> works with <meta>. In this case however, #UNWRAP# and #NOUNWRAP# are replaced by the unwrap attribute on the <mm:cdata> tag.

- **unwrap="**true | false"

  If not present, this attribute will default to false

Examples

EXAMPLE USING JSP TAGLIB	EXAMPLE USING MMXHTML

<!-- WebLogic Mobility Server will not attempt to parse this Javascript inside the **<mm:cdata>** tags, but this code will be processed by the browser. -->
<mm:meta name="MIS-CDATA-Control" content="Unwrap"/>
<script>
<mm:cdata unwrap="true">
function compareIt(var1,var2) {
if (var1 <= var2) then {
return 0
}
else {
return 1
}
}
</mm:cdata>
</script>
	<!-- The tag syntax for the same code example using mmXHTML looks like this: -->
	<meta name="MIS-CDATA-Control" content="Unwrap"/>
	<script>
	<![CDATA[#UNWRAP#
	function compareIt(var1,var2) {
	if (var1 <= var2) then {
	return 0
	}
	else {
	return 1
	}
	}
	]]>
	</script>

See also

`meta`

mm-exclude, mm:exclude

Description

Acts as a container for elements to be excluded when specific devices are targeted.

Attributes

- **where**="…"

  The ***where*** attribute enables the author to specify content selection conditions for devices.

  Conditions are constructed by specifying device attribute names and allowable values, using comparison operators from the Python programming language. Multiple conditions can be combined to make complex expressions by using the logical “and”, “or” and “not” operators from Python.

  Note: The “where” clause quoted string must not contain line breaks.

  WebLogic Mobility Server also supports use of the Python .endswith( ), .startswith( ) and .find( ) methods for partial matching.

  Restrictions:

  - At least one condition must be specified
  - Attribute names must be valid as taken from the Device Repository
  - All String attribute values must be wrapped in single quotes

Example

```
<!-- The following example illustrates how an address line is excluded for a
group of devices. -->
<mm-group id="company_details">
   <mm-head id="hd_company_details">Company Details</mm-head>
   <mm-body id="bd_company_details" idref="hd_company_details">
      <p>ABC Company Ltd.</p>
      <mm-exclude where="DeviceName.find('Ericsson')> 0">
         <p>North Business Park, Circular Road,Dublin,Ireland </p>
      </mm-exclude >
      <p>Phone: 888-000-111</p>
      <p>Fax: 000-888-111</p>
   </mm-body>
</mm-group>
```

See Also

`mm-include`

mm-group, mm:group

Description

Used to explicitly organize content into logical sections. An <mm-group> may optionally contain one <mm-head> element and zero or more <mm-body> elements.

Attributes

- **id**="…"

  Assigns a unique ID to an instance of this element.

- **title**="…"

  Specifies a title to be assigned to the element.

- **accesskey**="assign | 0-9"

  This is an optional attribute that gives the author some control over access keys on phones that support this functionality. The value "assign" will defer control of access keys to the phone itself. Alternatively, a number from 0-9 can be assigned by setting the attribute to the desired value. Both of these settings can be overridden if an <mm-structure> which contains a reference to this group has its *accesskeycontro*l attribute set to "removeall" in which case no access keys will be assigned. If the <mm-structure> has its *accesskeycontrol* attribute set to "assignall", access keys that have been assigned numbers will be overridden and the default settings of the phone will be used.

- **navstyle**="…"

  This attribute is used for styling navigational menus for PDAs and menu-driven devices.

Example

```
<!-- This example illustrates the use of the mm-group, mm-head and mm-body
elements. -->
<mm-group id="gp_company_details" title="Company Details">
 <mm-head id="hd_company_details">Company Details</mm-head>
 <mm-body id="bd_company_details" idref="hd_company_details">
  <p>ABC Company Ltd.</p>
</mm-body>
</mm-group>
```

See Also

mm-head, mm-body, mm-group-ref, mm-id-ref, mm-structure

mm-group-ref, mm:group-ref

Description

Used to reference an existing mm-group element. The content of the group can be displayed in full or as a hierarchical set of headings that the user can use to navigate to the appropriate section. The navigation hierarchy is typically used for menu-driven devices and PDAs. This is an empty tag, so it requires a slash (/) before the final angle bracket (>).

Attributes

- **idref**="…"

  The unique identifier associated with the <mm-group> element being referenced.

- **type**="options | normal"

  When content is being delivered to WML devices that support the Options menu, this attribute specifies whether the menu displays on the screen or whether it displays behind the **Options** button. This will have no effect on XHTML MP devices. Default is normal.

  Note: When type="options" the attribute display MUST be set to “headings”

- **depth**="flat | 0..9"

  Specifies the level of recursion to use when building the navigation hierarchy.

- **startdepth**="0-9"

  Specifies the level at which the navigation hierarchy starts. This attribute can only be used when display="headings"

- **display**="all | headings | links"

  Setting the value to “headings” displays the headings within a group down to the level (depth) specified. Setting the value to “all” displays everything within the group – headings, bodies and so on Setting the value to “links” displays headings and any links that occur within the group.

- **navstyle**="…"

  This attribute is used for styling navigational menus for PDAs and menu-driven devices.

Example

Depth	Display	Result
Flat	all	Displays the whole page as a flat structure.
0	headings	Displays a link to that group.
1	headings	Displays a collection of links to the group and its subgroups.
2	headings	Displays a collection of links to the group, its subgroups and its subgroup's subgroups.
0	all	Displays that group, both heading and body and a collection of links to its subgroups if any.

1	all	Displays that group, heading and body, its subgroups, heading, body and a collection of links to their subgroups if any.
0	links	Displays the heading of the group and any links that occur in the body of that group

See Also

`mm-group, mm-structure`

mm-head, mm:head

Description

The text inside the <mm:head> element can be used as a menu link or a page title on menu-driven devices and PDAs. Smaller screen devices often split content into multiple pages to accommodate memory and bandwidth restrictions. The heading is sometimes needed to give context to the piece of content that is being displayed. A maximum of one <mm:head> per group is permitted.

Attributes

- **id**="…"

  Assigns a unique ID to an instance of this element. If an <mm:body> element follows the head, its idref attribute must match the head's id attribute.

- **useradded**="yes | no"

  Indicates whether the text wrapped by the tag is already part of the content (="no") or has been specifically added (="yes"). Typically, you will add a head when the group is targeting menu-driven devices, so that the head becomes a link or a “card” title. Text that has been specifically added for this purpose will not appear on other devices.

Example

```
<!-- This example illustrates the use of the mm-group, mm-head and mm-body
elements -->
<mm-group id="gp_company_details">
  <mm-head id="hd_company_details">
       Company Details
   </mm-head>
   <mm-body id="bd_company_details" idref="hd_company_details">
      <p>ABC Company Ltd.</p>
   </mm-body>
</mm-group>
```

See Also

mm-group, mm-body, mm-group-ref

mm-id-ref, mm:id-ref

Description

This element causes WebLogic Mobility Server to place an instance of the referenced content at the point of insertion. This element is used in layout files. This is an empty tag, so it requires a slash (/) before the final angle bracket (>).

Attributes

- **idref**="…"

  The unique ID of the group being referenced. This can be an implicit group, such as a table or form, which has been assigned an id attribute, or an explicit group created with `<mm-group>`.

Example

```
<%-- This is a part of a layout file that is using PDA pagination to create a
navigation menu for PDAs. The mm:id-refs are being used to create a persistent
header and footer content which appears on all pages. They refer to predefined
groups of content whose id attribute values are 'header' and 'footer'.--%>
<mm:id-ref idref="header"/>
<mm:structure id="str2" where="IsPDA">
   <mm:group-ref idref="groupA" depth="0" display="headings" type="normal"/>
   <mm:group-ref idref="groupB" depth="0" display="headings" type="normal"/>
   <mm:group-ref idref="groupC" depth="0" display="headings" type="normal"/>
   <mm:group-ref idref="groupD" depth="0" display="headings" type="normal"/>
</mm:structure>
<mm:id-ref idref="footer"/>
```

See Also

`mm-group, mm-group-ref, mm-layout`

mm-img, mm:img

Description

Used to deliver the correct image based on the capabilities of the target device. Typically, multiple images will be placed inside a media-group element so that different image formats are available to be delivered depending on the requesting device type. The first image matching the criteria of this tag's where attribute is delivered; all others are ignored. Attribute of the XHTML <img> tag can also be used with this tag. This is an empty tag, so it requires a slash (/) before the final angle bracket (>).

Attributes

- **where**="…"

  The where attribute enables the author to specify content selection conditions for devices.

  Conditions are constructed by specifying device attribute names and allowable values, using comparison operators from the Python programming language. Multiple conditions can be combined to make complex expressions by using the logical “and”, “or” and “not” operators from Python.

  Note: The “where” clause quoted string must not contain line breaks.

  WebLogic Mobility Server also supports use of the Python .endswith( ), .startswith( ) and .find( ) methods for partial matching.

  Restrictions:

  - At least one condition must be specified
  - Attribute names must be valid as taken from the Device Repository
  - All String attribute values must be wrapped in single quotes

- **fittoscreen**="true | false"

  If this attribute is set to "true", the image width is resized to the UsableWidthPixels value of the device, as defined in the Device Repository. The image height is resized by the same factor so that the image maintains the same aspect ratio. This has an effect only if the original image is wider than the targeted screen width.

Example

```
<!—The WebLogic Mobility Server will select an image depending on the graphic
support of the requesting device. If the device matches more than one where
clause, it will receive the image from the first one that it matches. If no
match occurs, the text from the media-group's alt attribute will be sent to the
device.-->
<mm-media-group alt="No image provided">
   <mm-img where="ImgJpgProgressiveSupported" height="80" width="60" alt="Gold"
    src="img/gold.jpg"/>
   <mm-img where="ImgGIFSupported" height="40" width="30" alt="Gold"
    src="img/gold.jpg"/>
   <mm-img where="ImgWBMPSupported" height="16" width="12" alt="Gold" src="
    img/gold.wbmp"/>
</mm-media-group>
```

See Also

`mm-media-group, mm-logo`

mm-include, mm:include

Description

Acts as a container for elements that are to be included when a specific device class is targeted.

Attributes

- **where**="…"

  The where attribute enables the author to specify content selection conditions for devices.

  Conditions are constructed by specifying device attribute names and allowable values, using comparison operators from the Python programming language. Multiple conditions can be combined to make complex expressions by using the logical “and”, “or” and “not” operators from Python.

  Note: The “where” clause quoted string must not contain line breaks.

  WebLogic Mobility Server also supports use of the Python .endswith( ), .startswith( ) and .find( ) methods for partial matching.

  Restrictions:

  - At least one condition must be specified
  - Attribute names must be valid as taken from the Device Repository
  - All String attribute values must be wrapped in single quotes

Example

```
<!-- This example illustrates how to insert a line break into an address when
certain devices are targeted. -->
<mm-body id="bd_company_details" idref="hd_company_details">
   <p>ABC Company Ltd.</p>
   <p>North Business Park,
   <mm-include where="DeviceName.startswith('Nokia')"><br/></mm-include>
      Circular Road,
   <mm-include where="DeviceName.startswith('Nokia')"><br/></mm-include>
      Dublin,
   <mm-include where="DeviceName.startswith('Nokia')"><br/></mm-include>
      Ireland
   </p>
   <p>Phone: 888-000-111</p>
   <p>Fax: 000-888-111</p>
</mm-body>
```

See Also

`mm-exclude`

mm-layout, mm:layout

Description

Used to specify which layout files should be applied to a request page. Layout files are essentially XHTML or JSP files that contain a description of how output content is best arranged for a specific device or for certain classes of devices. This tag is placed inside the `<head>` element. It is an empty tag, so it requires a slash (/) before the final angle bracket (>).

Attributes

- **where**="…"

  The where attribute enables the author to specify content selection conditions for devices.

  Notes

  - where="IsFullBrowser" is not allowed with this tag
  - The “where” clause quoted string must not contain line breaks
  - Conditions are constructed by specifying device attribute names and allowable values, using comparison operators from the Python programming language. Multiple conditions can be combined to make complex expressions by using the logical “and”, “or” and “not” operators from Python.
  - WebLogic Mobility Server also supports use of the Python .endswith( ), .startswith( ) and .find( ) methods for partial matching.

  Restrictions

  - At least one condition must be specified
  - Attribute names must be valid as taken from the Device Repository
  - All String attribute values must be wrapped in single quotes

- **src="**url**"**

  A URL that points WebLogic Mobility Server to the layout file.

Example

```
<!-- The following example targets a distinct type of device with a specific
layout. -->
<?xml version='1.0'?>
<!DOCTYPE html PUBLIC "-//MOBILEAWARE//DTD MMXHTML 1.2//EN"
"http://www.mobileaware.com/DTD/mmxhtml_1.2.dtd">
<html>
<head>
   <title>Example Heading</title>
   <mm-layout where="IsPortraitPDA" src="layout_1.htm"/>
</head>
<body>
  <mm-group id="text1">
    <p> When this page is delivered to a portrait PDA, it will be formatted
        according to the style set out in the file layout_1.htm.</p>
  </mm-group>
</body>
</html>
```

See Also

`mm-id-ref, mm-group-ref`

mm-li, mm:li

Description

This tag defines a list item in a navigation list. It must be contained by <mm:nl> tags. It allows authors to create and style navigation lists that can specifically target mobile devices.

Attributes

- **navstyle**="…"

  This attribute is used for styling navigational menus. It is used here primarily with the nav-image property to reference an image to be used as part of the navigation, as can be seen in the following example. Additional styling for list items can be achieved using external style sheets.

- **href**="url"

  Contains the URL of the link created

Example

```
<!-- This example creates a styled navigation list with 4 elements.-->

<div style="border: 1px solid">
   <mm:nl navstyle="nav-format: list; nav-list-item-display: block">
      <mm:li navstyle="nav-image: url(dog.gif)" href="dogs.htm>Dogs</mm:li>
      <mm:li navstyle="nav-image: url(fish.gif)" href="fish.htm">Fish</mm:li>
      <mm:li navstyle="nav-image: url(cat.gif)" href="cat.htm">Cats</mm:li>
      <mm:li navstyle="nav-image: url(fish.gif)" href="fish.htm">Fish</mm:li>
   </mm:nl>
</div>
```

See Also

`mm:nl`

mm-logo, mm:logo

Description

Used to specify a logo image for WML devices. Logos typically appear for a user-specified interval before disappearing and being replaced by the main content.

Attributes

- **id**="…"

  Assigns a unique ID to an instance of the mm-logo element.

- **displaymode**="once | always"

  Allowed values are "once" or "always" indicating whether a logo should be displayed once per session or for every request.

- **period**="..."

  The value of this attribute is an integer, where the integer represents the number of seconds that the logo will be displayed.

Example 1

```
<!-- The following code snippet causes a company logo to appear for 2 seconds
before being replaced by the main content when the page is delivered to a WML
device. For each user session, this logo will appear only the first time the
user visits the page. -->
<mm-logo id="logo" displaymode="once" period="2">
   <img src="/companylogo.gif" alt="ABC Company Ltd."/>
</mm-logo>
```

Example 2

```
<!-- Images can be placed inside a media-group using the <mm-img> tag, so that
the best image is delivered to the requesting device.-->
<mm-logo id="logo" displaymode="once" period="2">
   <mm-media-group alt="Logo Image">
      <mm-img where="ImgGIFSupported" src="/companylogo.gif" alt="ABC Company
Ltd."/>
      <mm-img where="ImgWBMPSupported" src="/companylogo.wbmp" alt="ABC Company
Ltd."/>
   </mm-media-group>
</mm-logo>
```

See Also

`mm-img`

mm-media-group, mm:media-group

Description

Contains the <mm:img> elements that specify alternative media to deliver when a specific device is targeted. WebLogic Mobility Server will select an image from a media-group depending on the graphic support of the requesting device. If the device matches more than one where clause, it will receive the image from the first one that it matches. If no match occurs, the text from the media-group's *alt* attribute will be sent to the device.

Attributes

- **id**="…"

  Assigns a unique ID to an instance of this element. Used for navigation styling. This ID is referenced by a group-ref or structure tag to identify the image needed for the navigation.

- **alt**="…"

  Alternative text to be displayed in the event that no mm-img is provided for the requesting device. Do not supply an alt attribute if you do not want any text to appear if the requesting device is not on the list of devices in the <mm-img> tags.

- **style**="display:none"

  Optional attribute used for styling navigational menus for PDA and menu-driven devices. The value “display:none” renders the media-group invisible at the location that it displays in the content mark-up. WebLogic Mobility Server uses the media-group during the creation of navigation menu styles that use images.

Example

```
<!-- This example shows a partial layout file which contains two media-groups
used for styling navigation menus with images. The group-ref tags in the
structure identify the desired image to appear with the menu link by referring
to the media-group's id. To ensure that the image does not appear anywhere in
the delivered content except in the navigation, the media-group's style
attribute is set to "display:none".-->
<body>
   <mm:media-group id="car" style="display:none" alt="*">
      <mm:img where="ImgGIFSupported" src="img/car.gif" alt="*"/>
      <mm:img where="ImgWBMPSupported" src="img/car.wbmp" alt="*"/>
   </mm:media-group>
   <mm:media-group id="boat" style="display:none" alt="*">
      <mm:img where="ImgGIFSupported" src="img/boat.gif" alt="*"/>
      <mm:img where="ImgWBMPSupported" src="img/boat.wbmp" alt="*"/>
   </mm:media-group>
   <mm:structure id="pagination_str" where="IsPDA">
      <mm:group-ref idref="groupA" depth="0" display="headings" type="normal"
navstyle="nav-image:url(#car)"/>
      <mm:group-ref idref="groupB" depth="0" display="headings" type="normal"
 navstyle="nav-image:url(#boat)"/>
   </mm:structure>
< /body>
```

See Also

mm-img, mm:group-ref

mm-nl, mm:nl

Description

This tag name comes from the words “navigation list”. It allows authors to create and style navigation lists that can specifically target mobile devices.

Attributes

- **navstyle="…"**

  This attribute is used for styling navigational menus. The following values can be used: nav-format: [list | table], nav-list-item-display: [none | inline | block], nav-list-style-type: pipe, nav-image:[url(*#id* | *url*), none], nav-text-display: [none | inline | block], nav-table-columns:[*int*], nav-table-rows[*int*]. Additional styling can be achieved using external style sheets.

- **where**="…"

  The where attribute enables the author to specify content selection conditions for devices.

  Notes

  - where="IsFullBrowser" is not allowed with this tag
  - The “where” clause quoted string must not contain line breaks
  - Conditions are constructed by specifying device attribute names and allowable values, using comparison operators from the Python programming language. Multiple conditions can be combined to make complex expressions by using the logical “and”, “or” and “not” operators from Python
  - WebLogic Mobility Server also supports use of the Python .endswith( ), .startswith( ) and .find( ) methods for partial matching

  Restrictions

  - At least one condition must be specified
  - Attribute names must be valid as taken from the Device Repository
  - All String attribute values must be wrapped in single quotes

Example

```
<!-- This example creates a styled navigation list with 4 elements.-->
<div style="border: 1px solid">
   <mm:nl navstyle="nav-format: list; nav-list-item-display: block">
      <mm:li navstyle="nav-image: url(dog.gif)" href="dogs.htm">Dogs</mm:li>
      <mm:li navstyle="nav-image: url(fish.gif)" href="fish.htm">Fish</mm:li>
      <mm:li navstyle="nav-image: url(cat.gif)" href="cat.htm">Cats</mm:li>
      <mm:li navstyle="nav-image: url(fish.gif)" href="fish.htm">Fish</mm:li>
   </mm:nl>
</div>
```

See Also

`mm:li`

mm:page

Description

This is a required tag for content that uses the WebLogic Mobility Server JSP taglib. The <mm:page> tag wraps the page content. Setting attributes in this tag can enhance the performance of JSP page processing by allowing the author the choice of bypassing the WebLogic Mobility Server transformation engine and/or of turning off the mixed-mode tag checking feature.

Attributes

- **content**="true | false"

  Allows the author to turn on or off the mixed-mode tag checking. If the author knows that the page only contains the WebLogic Mobility Server JSP taglib tags (mm:) and no mmXHTML tags (mm-), this attribute can be turned off to improve performance. The default is true.

- **bypass**="true | false"

  Allows the author to bypass the WebLogic Mobility Server transformation engine. This might be done to improve performance for pages that are to be delivered exclusively to a PC. The default is false.

Example

```
 <!— The WebLogic Mobility Server will not check this file for mixed tags. -->
<%@ taglib uri="mmJSPtaglib" prefix="mm" %>
<mm:page content="false">
<html>
<head>
   <title>Skip Tag Check</title>
</head>
<body>
   <p>This page has WebLogic Mobility Server JSP tags exclusively. Tag
      checking can be turned off.</p>
</body>
</html>
</mm:page>
```

mm-phone-number, mm:phone-number

Description

Used to include a dialable phone number link in content delivered to WTAI enabled devices.

Attributes

- **num**="…"

  Specifies a valid dialable phone number.

- **cti**="…"

  Used when the requesting device is an iMode device. This attribute lets you create a link that will be dialed when the user selects the link. Unlike the num attribute, cti gives the user the option of including pauses, characters and extension numbers after the main phone number. The hash mark (#) and the asterisk (*) are also supported.

Example

```
<!--This example will create a link that, when clicked, will dial +35312410500.
If the phone is an iMode device, it will use the value of the cti attribute.
After dialing the number, once the receiving end has picked up, it will pause
for 2 seconds before dialing the extension 538#. -->
<mm:phone-number num="+ 35312410500" cti="+35312410500/,,538#">
     Call Julia.
</mm:phone-number>
```

mm-structure, mm:structure

Description

Used to determine which groups are to be delivered to menu-driven devices or to create navigation menus for PDAs. The <mm:structure> element contains <mm:group-ref> elements which refer to the groups of content that you want delivered. Typically, this element is placed in a layout file.

Attributes

- **id**="…"

  Assigns a unique ID to an instance of this element.

- **where**="…"

  The where attribute enables the author to specify content selection conditions for devices.

  Notes

  - where="IsFullBrowser" is not allowed with this tag
  - The “where” clause quoted string must not contain line breaks
  - Conditions are constructed by specifying device attribute names and allowable values, using comparison operators from the Python programming language. Multiple conditions can be combined to make complex expressions by using the logical “and”, “or” and “not” operators from Python
  - WebLogic Mobility Server also supports use of the Python .endswith( ), .startswith( ) and .find( ) methods for partial matching.

  Restrictions

  - At least one condition must be specified
  - Attribute names must be valid as taken from the Device Repository
  - All String attribute values must be wrapped in single quotes

- **accesskeycontrol**="assignall | assignempty | removeall"

  This is an optional attribute that gives the author some control over access keys on phones that support this functionality. The value "assignall" will defer control of access keys to the phone itself. This applies to all the groups that are being referenced by the mm-structure regardless of whether or not the groups themselves have been explicitly assigned a number. The value "assignempty" will defer control of access keys to the phone only in the case where a group within the mm-structure has not explicitly been assigned a number. Setting accesskeycontrol="removeall" will disallow any accesskeys being set thereby overriding any accesskey settings within the individual groups contained in the `<mm-structure>`. The value "removeall" will not remove accesskey settings in other XHTML tags such as `<input accesskey="...">` or `<a accesskey="...">`.

  Note: The attribute values “assignall” and “assignempty” are not supported on iMode phones.

- **navstyle**="…"

  This attribute is used for styling navigational menus for PDAs and menu-driven devices.

- **idheader**="…"

  This attribute is used to specify the mm-group that you want to use as a header on each page presented on a menu-driven mobile device.

- **idfooter**="…"

  This attribute is used to specify the mm-group that you want to use as a footer on each page presented on a menu-driven mobile device.

Example

```
<!--  Groups referenced from within this mm-structure element will be displayed
on menu-driven browsers. -->
<mm-structure id="structure_1" where="IsMenuDriven">
   <mm-group-ref idref="group_1" type="normal" depth="flat" display="all"/>
   <mm-group-ref idref="group_2" type="normal" depth="0" display="headings"/>
</mm-structure>
```

See Also

`mm-group-ref, mm-group`

mm-table-model, mm:table-model

Description

Use the `<mm:table-model>` element to enable WebLogic Mobility Server to transform tables for small devices. Table information can be presented differently depending on the capabilities of the requesting device. Data can be “flattened” for devices that do not support tables, or for devices that have screen widths that would cause the data to wrap excessively. Multiple table-models can be used to tailor the data presentation to the capabilities of the requesting device. If the device matches more than one table-model where clause, the first one that matches will be used to structure the table. The table-model tag should be placed directly below the XHTML `<table>` tag. If there are nested tables, a separate table-model tag is required for each nested table. This is an empty tag, so it requires a slash (/) before the final angle bracket (>).

Attributes

- **title**="..."

  This is an optional attribute. If present, it forms the table title when used with the tabletype="group". If title is not present when tabletype="group" and a link name is required, the keyword “Data” will be used instead.

- **major**="row | column"

  Determines whether the data is presented row first, or column first. The default is row first.

- **headlocation**="…"

  Identifies which row (or column) number should be used to identify the heading for each section.

- **bodylocation**="…"

  This optional attribute specifies which rows or columns are to be displayed. Without this attribute explicitly defined, WebLogic Mobility Server will default to displaying the entire table although a warning will appear in the application server console. To explicitly direct WebLogic Mobility Server to display the entire table, set bodylocation="*". Part of the table can be displayed by setting this attribute to a space-separated list of numbers representing the rows (or columns) to be sent to the device.

- **tabletype**="normal | group"

  If the required attribute tabletype is set to normal, WebLogic Mobility Server will attempt to display the entire table. If the table is too big for WebLogic Mobility Server to fit onto a “card”, the page will be broken into multiple cards when necessary.

  Setting tabletype to group causes the table headers to be rendered as links. These links can be navigated to view detailed table content presented in a "table header: table data" pairing. Using tabletype="group" for a device that can render tables will have little effect unless the table is first “flattened” using sdtransform**.**

- **sdtransform**="base-transform | on-same-card |on-new-card"

  - “base-transform” is an optional attribute used to "flatten" tables that are sent to menu-driven devices or PDAs that support tables. This may be necessary to avoid excessive wrapping that could occur if the device were to render the data as a table on a small screen. Table content is presented as a series of "table header: table data" pairings. If the attribute is not present, the result will depend on the table support provided by the device. Using this attribute will have no impact on non-table-supporting devices.

  - “on-same-card” is an optional attribute used to display new tables on the same card.

    Note: This will not create a link to that nested table

  - “on-new-card” is an optional attribute used to create a link to a nested table that is placed on a separate card

- **where**="…"

  The where attribute enables the author to specify content selection conditions for devices.

  Notes

  - where="IsFullBrowser" is not allowed with this tag

  - The “where” clause quoted string must not contain line breaks

  - Conditions are constructed by specifying device attribute names and allowable values, using comparison operators from the Python programming language. Multiple conditions can be combined to make complex expressions by using the logical “and”, “or” and “not” operators from Python

  - WebLogic Mobility Server also supports use of the Python .endswith( ), .startswith( ) and .find( ) methods for partial matching.

  Restrictions

  - At least one condition must be specified

  - Attribute names must be valid as taken from the Device Repository

  - All String attribute values must be wrapped in single quotes

Example

```
<!-- Two table-model tags have been placed inside the XHTML table in the
example. The first one is responsible for transforming tables being delivered
to devices that are both capable of rendering tables and have a screen width
greater than or equal to 196 pixels. The second one is responsible for
transforming tables being delivered to devices that either cannot render tables
or have a screen width less than 196 pixels. -->
<table border="1">
   <mm:table-model headlocation="1" bodylocation="*" major="row"
    tabletype="normal" where="UAProf.BrowserUA.TablesCapable and
    UsableWidthPixels >= 196"/>
   <mm:table-model headlocation="1" bodylocation="*" major="row"
    sdtransform="base-transform" tabletype="normal" where="(not
    UAProf.BrowserUA.TablesCapable) or (UAProf.BrowserUA.TablesCapable and
    (UsableWidthPixels < 196))"/>
   <tr><td>June</td>
       <td>July</td>
       <td>August</td>
   </tr>
```

```
   <tr><td>Swimming</td>
       <td>Kayaking</td>
       <td>Rock Climbing</td>
   </tr>
</table>
```

Appendix B – Mobility Delivery Context API

Introduction

The Mobility Delivery Context API provides a mechanism for developers to add “device-aware” logic into their applications. The Mobility Delivery Context API provides methods to request a delivery context and subsequently access device attributes in the retrieved delivery context.

Applications can use the Mobility Delivery Context API to:

- Retrieve a delivery context for the device currently interacting with the application by passing in the associated http request object. These are referred to as In-session Queries because WebLogic Mobility Server automatically maintains a delivery context for the current device in the session.
- Retrieve delivery contexts for one or more devices that match conditions passed by the application as parameters in the delivery context query. These are referred to as Out-of-session Queries.

Note: The Mobility Delivery Context API provides access to the UAProf device attributes available using the JSR 188 Delivery Context API and also provides access to the Mobility proprietary device profile attributes that are not available using the JSR 188 Delivery Context API.

Note: The JSR 188 Delivery Context API has not yet been extended to support Out-of-session Queries.

Requesting the Delivery Context for an Active HTTP Session

The method call used to retrieve the Delivery Context for the device associated with the active HTTP Session is

```
DeliveryContextFactory.getDeliveryContext (request)
```

where request is the request object.

The following code block illustrates how this method is used to request a Delivery Context for a device associated with an active HTTP Session.

```
<%@ taglib uri="mmJSPtaglib" prefix="mm" %>
<mm:page content="false">
<%@ page import="com.mobileaware.deliverycontext.*" %>
<html>

<head>
   <title>MA Delivery Context API</title>
   <mm:structure id="st_101" where="IsMenuDriven">
      <mm:group-ref idref="gp_101" depth="flat" type="normal" display="all"/>
   </mm:structure>
</head>
<body>
<mm:group id="gp_101" title="Details">
<%
DeliveryContext dc = DeliveryContextFactory.getDeliveryContext(request);
try {
String deviceName= dc.getAttribute("DeviceName");
String imgList= dc.getAttribute("ImgTypePref");
%>
  <p>I am a <%=deviceName%>.</p>
  <p>I accept images that are: <%=imgList%></p>
```

```
<%
} catch(Throwable e) {
  out.println("<p>Problem creating delivery context.</p>");
}
%>
</mm:group>
</body>
</html>
</mm:page>
```

The results when accessing Mobility attributes of a WML emulator are shown here.

Results

Image Courtesy of Openwave Systems Inc

Request Out-of-Session Delivery Contexts

Establish the Delivery Context Store

A Delivery Context Store must be established before attempting to retrieve Delivery Contexts. This Delivery Context Store indicates the Device Repository instance against which the subsequent Delivery Context queries should be issue. There are two methods for establishing the Delivery Context Store depending on whether the Device Repository is deployed in a relational database or as an XML file.

- To establish a Delivery Context Store where the Device Repository is deployed in a database, the following method is used:

  ```
  DeliveryContextStoreFactory.getDeliveryContextDBStore("deviceDB.url",
  "deviceDB.driver")
  ```

  where `deviceDB.url` is the location of the Device Repository (including user and password if required) and `deviceDB.driver` is the driver to be used to access the database. These database connection parameters should be provided by the database administrator.

- To establish a Delivery Context Store where the Device Repository is deployed as an XML file, the following method is used:

  ```
  DeliveryContextStoreFactory.getDeliveryContextXMLStore("deviceDB.url")
  ```

  where `deviceDB.url` is the path to the deployed Device Repository XML file.

Methods for Retrieving Delivery Contexts

There are four method calls that can be used to retrieve Delivery Contexts for devices in the Device Repository. The methods are described as follows:

- To retrieve a single device where a specified attribute matches a specified value:

  ```
  DeliveryContextFactory.getDeliveryContext(deliveryContextStore,
  "attributeName", "attributeValue")
  ```

  where `deliveryContextStore` indicates the established delivery context store to query, `attributeName` indicates the device store attribute to match on, and `attributeValue` indicates the value of the indicated Device Repository attribute to match on. For example, to retrieve a delivery context for the Sony Ericsson P900 you could use the following:

  ```
  DeliveryContext deliveryContext =
  DeliveryContextFactory.getDeliveryContext(deliveryContextStore,
  "DeviceUniqueName","root^xhtmlmp^ericsson(xhtml)^sonyericssonp900")
  ```

  Note: If more than one device matches the indicated criteria, the first device that matched will be returned.

- To retrieve a single device matching a specified "where" expression:

  ```
  DeliveryContextFactory.getDeliveryContext(deliveryContextStore,
  "whereExpression")
  ```

  where `deliveryContextStore` indicates the established delivery context store to query and `whereExpression` indicates the where expression to evaluate and return the first matching device for. For example, an alternative way to retrieve a delivery context for the Sony Ericsson P900 would be:

```
DeliveryContext deliveryContext =
DeliveryContextFactory.getDeliveryContext(deliveryContextStore,
"DeviceUniqueName==’root^xhtmlmp^ericsson(xhtml)^sonyericssonp900’")
```

Note: If more than one device matches the indicated criteria, the first device that matched will be returned.

- To retrieve a delivery context for ALL devices where a specified attribute matches a specified value:

  ```
  DeliveryContextFactory.getDeliveryContexts(deliveryContextStore,
  "attributeName","attributeValue")
  ```

  where `deliveryContextStore` indicates the established delivery context store to query, `attributeName` indicates the device store attribute to match on, and `attributeValue` indicates the value of the indicated Device Repository attribute to match on. For example, to retrieve a delivery context for all devices profiled as PDAs you could use the following:

  ```
  DeliveryContext[] deliveryContexts =
  DeliveryContextFactory.getDeliveryContexts(deliveryContextStore,
  "isPDA","true")
  ```

- To retrieve a delivery context for ALL devices matching a specified “where” expression:

  ```
  DeliveryContextFactory.getDeliveryContexts(deliveryContextStore,
  "whereExpression")
  ```

  where `deliveryContextStore` indicates the established delivery context store to query and `whereExpression` indicates the where expression to evaluate and return all matching devices for. For example, an alternative way to retrieve a delivery context for all PDA devices would be:

  ```
  DeliveryContext[] deliveryContexts =
  DeliveryContextFactory.getDeliveryContexts(deliveryContextStore,"IsPDA")
  ```

- Once the DeliveryContextStore is no longer needed it should be destroyed to free up any resources it may be holding onto (for example, Database connections):

  ```
  deliveryContextStore.destroy();
  ```

The following code block illustrates the use of these methods to establish a Delivery Context Store and subsequently request a Delivery Context for the SonyEricsson P900.

```
import java.sql.*;
import com.mobileaware.deliverycontext.*;

public final class DeliveryContextExternalQuery{
   public static void main(String args[]){
DeliveryContextStore deliveryContextStore =
DeliveryContextStoreFactory.getDeliveryContextDBStore(
    "jdbc:mysql://test7/madb?user=root&password=",
           "org.gjt.mm.mysql.Driver");

       try{
             //Gets the matching device whose Unique Name is root^wml
DeliveryContext deliveryContext1 =  DeliveryContextFactory.getDeliveryContext(
                deliveryContextStore,
                "DeviceUniqueName",
                "root^xhtmlmp^ericsson(xhtml)^sonyericssonp900");

System.out.println("DeviceUniqueName: " +
deliveryContext1.getAttribute("DeviceUniqueName"));
```

```
System.out.println("ImgGIFSupported: " +
deliveryContext1.getAttribute("ImgGIFSupported"));
System.out.println("IsPDA: " + deliveryContext1.getAttribute("IsPDA"));
System.out.println("CharsetSupported: " +
deliveryContext1.getAttribute("UAProf.SoftwarePlatform.CcppAccept-Charset"));

             //Gets the first matching device that is a PDA
DeliveryContext deliveryContext2 = DeliveryContextFactory.getDeliveryContext(
                 deliveryContextStore,
                 "IsPDA");

System.out.println("DeviceUniqueName: " +
deliveryContext2.getAttribute("DeviceUniqueName"));
System.out.println("ImgGIFSupported: " +
deliveryContext2.getAttribute("ImgGIFSupported"));
System.out.println("IsPDA: " + deliveryContext2.getAttribute("IsPDA"));
System.out.println("CharsetSupported: " +
deliveryContext2.getAttribute("UAProf.SoftwarePlatform.CcppAccept-Charset"));

             //Gets all devices that are PDAs
DeliveryContext deliveryContexts[] =
DeliveryContextFactory.getDeliveryContexts(
                 deliveryContextStore,
                 "IsPDA");

System.out.println("DeviceUniqueName: " +
deliveryContexts[2].getAttribute("DeviceUniqueName"));
System.out.println("ImgGIFSupported: " +
deliveryContexts[2].getAttribute("ImgGIFSupported"));
System.out.println("IsPDA: " + deliveryContexts[2].getAttribute("IsPDA"));
System.out.println("CharsetSupported: " +
deliveryContexts[2].getAttribute("UAProf.SoftwarePlatform.CcppAccept-
Charset"));

       }catch(AttributeValueUndefinedException e){
              System.out.println(e.getMessage());
       }finally{
              deliveryContextStore.destroy();
       }

    }
}
```

Note: The DeliveryContextStore is responsible for caching and connection pooling and therefore is expensive to create. It is recommended that this Object be stored and reused for connection to the database for as long as possible. It is also necessary to call `destroy()` on this object when you are finished as it needs to free up database connections. The following JSP example demonstrates the recommended practice of storing this within the scope of the servlet application.

```
DeliveryContextStore deliveryContextStore = (DeliveryContextStore)
 application.getAttribute("DeliveryContextStore");

if(deliveryContextStore==null) {
deliveryContextStore =
DeliveryContextStoreFactory.getDeliveryContextDBStore(
"jdbc:mysql://cleantest-dev/dbHEAD-G?user=root&password=",
"org.gjt.mm.mysql.Driver");

       application.setAttribute("DeliveryContextStore",deliveryContextStore);
}
```

Available Public Methods

There are a number of public methods that can be used to access attributes in the retrieved Delivery Context:

Public Methods

API Call	Attribute DataType	Java Type Returned
getIntAttribute()	Integer	int
getLiteralAttribute()	Literal	String
getbooleanAttribute()	Boolean	Boolean
getURIAttribute()	URI	String
getRationalAttribute()	Rational	double
getDimensionAttribute()	Dimension	Dimension
getSequenceAttribute()	Sequence	List
GetBagAttribute()	Bag	Set
getAttribute()	Integer	String
getAttribute()	Literal	String
getAttribute()	Boolean	String
getAttribute()	URI	String
getAttribute()	Rational	String
getAttribute()	Dimension	String
getAttribute()	Sequence	String
getAttribute()	Bag	String

The methods listed here have either one or two parameters: an attribute name or an attribute name and a default to return in the event that no value is available.

Example

```
GetIntAttributes("MaxWapDeckSize");
GetIntAttribute("MaxWapDeckSize", 1400);
```

Handle Exceptions

There are four exceptions that can be thrown when using the methods listed in the preceding table. It is considered good practice to place try – catch blocks around these method calls.

DeliveryContextRuntimeException

Thrown when an unexpected condition causes DeliveryContext creation or method to fail. The exception message describes the problem, such as "invalid ServletRequest Object".

InvalidAttributeTypeException

Thrown when a type-specific method is called on an attribute of the wrong type. For example, using getBooleanAttribute() to obtain a String value.

NoSuchAttributeException

Thrown when an attribute name is requested and that attribute does not exist. This is different from the attribute existing and not having a value defined for the requesting device.

AttributeValueUndefinedException

Thrown when an attribute name is requested but the attribute value does not exist.

Appendix C – Deprecated Items

This section describes the deprecated items in this version of WebLogic Mobility Server and the features that replace those items.

The Device Repository has adopted the Composite Capabilities / Preferences Profile (CC/PP) standard. New attributes have been added to the Device Repository to conform to the standard. A number of the Mobility proprietary attributes have been replaced in the process.

The deprecated attributes are still supported although a warning will appear in the console window if they are used.

The following table lists the deprecated attributes and the attributes that have replaced them.

Deprecated Device Repository Attributes

Deprecated Attributes	CC/PP Replacement
ColorDepth	UAProf.HardwarePlatform.BitsPerPixel
Brand	UAProf.HardwarePlatform.Vendor
OSType	UAProf.SoftwarePlatform.OSName
OSVersion	UAProf.SoftwarePlatform.OSVersion
JavaScriptSupported	UAProf.BrowserUA.JavaScriptEnabled
TableSupported	UAProf.BrowserUA.TablesCapable
ScreenAspectRatioPixels	UAProf.HardwarePlatform.PixelAspectRatio
CharsetSupported	UAProf.SoftwarePlatform.CcppAccept-Charset
EmailClient	UAProf.SoftwarePlatform.Email-URI-Schemes
AcceptHeader	UAProf.SoftwarePlatform.CcppAccept
ColorType	UAProf.HardwarePlatform.ColorCapable
AudioFormatSupported	UAProf.SoftwarePlatform.CcppAccept
ImageFormatSupported	UAProf.SoftwarePlatform.CcppAccept
MultipartSupported	UAProf.SoftwarePlatform.CcppAccept
VideoSupported	UAProf.SoftwarePlatform.CcppAccept
SoundHandling	UAProf.SoftwarePlatform.CcppAccept
USSDSupported	UAProf.NetworkCharacteristics.SupportedBearers
WTLSSupported	UAProf.NetworkCharacteristics.SecuritySupport

FoundationProfile1xSupported	UAProf.SoftwarePlatform.JavaPlatform
CLDC1xSupported	UAProf.SoftwarePlatform.JavaPlatform
CDC1xSupported	UAProf.SoftwarePlatform.JavaPlatform
JavaPhone1xSupported	UAProf.SoftwarePlatform.JavaPlatform
MIDP1xSupported	UAProf.SoftwarePlatform.JavaPlatform
PersonalJava1xSupported	UAProf.SoftwarePlatform.JavaPlatform
WAPVersion	UAProf.WapCharacteristics.WapVersion
MIDP2xSupported	UAProf.SoftwarePlatform.JavaPlatform
WTAIMakePhoneCallSupported	UAProf.WapCharacteristics.WtaiLibraries
WTAIAddPhoneBookEntrySupported	UAProf.WapCharacteristics.WtaiLibraries
WAPPushSISupported	UAProf.PushCharacteristics.Push-Accept
WAPPushSLSupported	UAProf.PushCharacteristics.Push-Accept
WAPPushSupported	UAProf.PushCharacteristics.Push-Accept
ViewableHeight	UAProf.HardwarePlatform.Screensize
ViewableWidth	UAProf.HardwarePlatform.Screensize

Appendix D – Use the Generic Log Monitor Facility with Log4J

WebLogic Mobility Server provides a Generic Log Monitor facility allowing WebLogic Mobility Server Exceptions, Warnings, and Diagnostics to be delivered to an alternative Logging system in addition to the standard WebLogic Mobility Server logging mechanisms.

Note: This section describes how to use the GenericLogMonitor facility with Log4J. However, it is possible to use an alternative logging tool if required.

In order to use the GenericLogMonitor facility with Log4J it is necessary to:

- Create a class that implements this interface - this is the class that will intercept WebLogic Mobility Server Exceptions, Warnings and Diagnostics.
- Place this class in the com.mobility.diagnostics package under the **WEB-INF\classes** directory of the web application.
- Specify this class in the *mis.properties* file found in the **WEB-INF\classes** directory of the web application.
- Ensure the log4j jar file is in your class path (or **lib** directory of the webapp).
- Create a valid Log4J properties file that will be accessed from within the implementation class - this will detail where messages should be sent along with the format of how they should be displayed. For example messages could be sent to the console, a log file, a remote server or a combination thereof.
- If the class specified in the *mis.properties* file then messages will be outputted in default mode (to the console).

A summary of the required changes is captured in this table.

File to change	Required Changes

mis.properties	When subscribing to Diagnostics, you need to specify the class that will intercept Diagnostics messages and WebLogic Mobility Server Exceptions/Warnings in the *mis.properties* file (located in the **WEB-INF\classes** directory of the web application). For example:
	
	`diagnostics.startup.subscriptions.misLog.topic: MIS.Browser.Request`
	
	`diagnostics.startup.subscriptions.misLog.level: normal`
	
	`diagnostics.startup.subscriptions.misLog.classname: com.mobileaware.diagnostics.GenericLogMonitorImpl`

log4j.properties	You must configure log4j with appropriate settings based on the desired format of the captured logs.
	An example *log4j.properties* file that will log WebLogic Mobility Server messages as Log4J messages to both the console and a file in a specified format is shown below:
	`log4j.rootLogger=DEBUG, A1`
	
	`log4j.appender.A1=org.apache.log4j.ConsoleAppender`
	`log4j.appender.A1.layout=org.apache.log4j.PatternLayout`
	
	`log4j.appender.A1=org.apache.log4j.RollingFileAppender`
	`log4j.appender.A1.File=d:\\test.log`
	
	`log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c - %m%n`
	
	`log4j.logger.com.mobileaware.diagnostics.GenericLogMonitor =INFO`

GenericLogMonitor Implementation Class

It is necessary to include a class that implements the GenericLogMonitor interface – see the following interface description.

This class is responsible for intercepting the specified disgnostics in the *mis.properties* file.

This class must also be specified in the `diagnostics.startup.subscriptions.misLog.classname` property in the *mis.properties* file.

This class must be placed in the com.mobility.diagnostics package under the **WEB-INF\classes** directory of the web application.

GenericLogMonitor Interface

The publish() method in the GenericLogMonitor interface is responsible for publishing Exceptions, Warnings and Diagnostics to an external class:

```
void publish(String messageName, String message, boolean warnings, boolean
errors, long time, String requestURL, String webAppName, String sessionID,
String deviceName, String username, String localServerIP, String serverPort,
String localServerHostname, String requestRemoteHost, String
requestRemoteAddr);
```

A description of the information accessible via the parameters is provided in the following table.

Information Provided by the publish() Method

Parameter	Description
messageName	The message name or type - typically MIS.Warning, MIS.FatalException or Diagnostic topic
message	The message associated with the Exception, Warning or Diagnostics including the error/warning code if appropriate
warnings	True if the message is a warning, False otherwise
errors	True if the message is an Error, False otherwise
time	Date & Time Stamp indicating when the message was generated
requestURL	The URL being requested
webAppName	The name of the Web App
sessionID	The Session ID of the user's current session if applicable
deviceName	The device making the request (null until the device is recognized)
username	N/A
localServerIP	Server IP
serverPort	Server Port
localServerHostname	Server Hostname
requestRemoteHost	Remote Hostname
requestRemoteAddr	Remote IP Address

Sample Implementation of the GenericLogMonitor Interface

```
package com.mobileaware.diagnostics;

import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

/*
 * This is a test implementation of the GenericLogMonitor interface
 * that integrates with Log4J
 */
public class GenericLogMonitorImpl implements GenericLogMonitor{
       static Logger logger =
Logger.getLogger(GenericLogMonitorImpl.class.getName());

       public GenericLogMonitorImpl () {
              PropertyConfigurator.configure("D:/Log4J/myLog.properties");
       }

       /* This method intercepts any Exceptions, Warnings and Diagnostics and
maps them
        * to corresponding Log4J messages.
        * The mappings are matched in the following order:
        *      1. MIS.Warning -> Log4J message Warn
        *      2. MIS.FatalException -> Log4J message Fatal
        *      3. MIS logging of type error ->  Log4J message Error
        *      4. MIS logging of type warning ->  Log4J message Warn
        *      5. MIS logging not matched by the above -> Log4j message Info
        * These can then be filtered in the log4j properties file.
        */
       public void publish(String messageName, String message, boolean warnings,
boolean errors, long time, String requestURL, String webAppName, String
sessionID, String deviceName, String username, String localServerIP, String
serverPort, String localServerHostname, String requestRemoteHost, String
requestRemoteAddr) {

              if (messageName.equalsIgnoreCase("MIS.Warning") && errors){
                     logger.error("["+messageName+"] - " + message + " [Device:
" + deviceName + ", URL: " + requestURL + "]");
              }else if (messageName.equalsIgnoreCase("MIS.Warning")){
                     logger.warn("["+messageName+"] - " + message + " [Device: "
+ deviceName + ", URL: " + requestURL + "]");
              }else if (messageName.equalsIgnoreCase("MIS.FatalException")){
                     logger.fatal("["+messageName+"] - " + message + " [Device:
" + deviceName + ", URL: " + requestURL + "]");
              }else if (errors==true){
                     logger.error("["+messageName+"] - " + message + " [Device:
" + deviceName + ", URL: " + requestURL + "]");
              }else if (warnings==true){
                     logger.warn("["+messageName+"] - " + message + " [Device: "
+ deviceName + ", URL: " + requestURL + "]");
              }else{
                     logger.info("["+messageName+"] - " + message + " [Device: "
+ deviceName + ", URL: " + requestURL + "]");
              }
       }
}
```

Appendix E – FAQ

Question 1

On the first request, I am presented with the following error message in the Application Server Console:

"There is a problem with the database connection. This may be due to a network fault or an incorrect database setting in the WebLogic Mobility Server configuration file".

What is causing this problem?

Solution

Ensure that the Application Server on which WebLogic Mobility Server is running and the Database Server is both running and connected to the network.

The most likely cause for a database problem is incorrect database settings in the WebLogic Mobility Server configuration file. This file is called *mis.properties* and is found in the **WEB-INF/classes/** folder in the **webapps** directory containing the file you are requesting.

Ensure that:

- the deviceDB.url is set to a valid URL for that database server
- the deviceDB.driver has a correct driver (and correctly spelled) specific to the database server you are using
- the devceDB.username and deviceDB.password are correct

If this does not solve the problem, then please contact your Database Administrator to ascertain if the problem is with the Database Server.

Question 2

Why do my error pages not appear on WAP?

Solution

WAP Gateways/Phones often catch the HTTP status and turn it into a generic message. This can happen also in IE/Opera if you enable "friendly HTTP error pages" - a setting enabled by default in both of these browsers.

Question 3

Why are there are incorrect or missing characters in the delivered content?

Solution

You may not have specified the correct content encoding in the source file. Sometimes WebLogic Mobility Server will translate the page during transformation, for example windows-1252 -> utf-8 in the case of WML transformation. If you haven't specified the correct encoding in the file, there is the possibility that WebLogic Mobility Server may fail to correctly detect the encoding of the file. As a result, certain characters will be translated into the incorrect byte representation for the target encoding. Specify the encoding in either: (a) the HTTP header (b) the XML declaration in the marked up file or (c) the `<meta>` tag in the marked up file. See also “Work with Character Sets” on page 112 in this manual.

Question 4

When I request a page from WebLogic Mobility Server, I receive a short message on my web browser indicating that the following error occurred:

```
Internal error [40010u]
```

In addition, the following error message displays on the Server Console window:

```
*[MIS.FatalException] content processing exception building idocument
[40010]:com.mobileaware.mobilitycontrol.MMParseException: SAXException
processing: null [60040]( Time:1037190843650,
SessionID:1BB7FCD73D36AF9BFA74436AEBD3C8B3, Username:null,
ServerIP:199.199.199.199, ServerPort:8080, ServerHostname:199.199.199.199,
RequestRemoteHost:MYHOST, Errors:true, RequestRemoteAddr:199.199.199.199,
RequestURL:/myapp/index.jsp, deviceName:root^html^mozilla/4, webAppName:/myapp)
```

Solution

This problem is most commonly caused by mistakes in the content that is being processed by WebLogic Mobility Server. Check for the following common errors:

Presence of special characters: certain characters such as (&, <, >) have special meaning in XHTML/XML. If these characters are used incorrectly, it can cause WebLogic Mobility Server to fail in the transformation of the content.

The following table will help guide you in replacing the characters with their correct entity reference.

Entity References

Character	Entity
&	&amp;
<	<
>	>

Note: WebLogic Mobility Server does not normally tolerate single ampersand characters (aside from the entity reference) within the page, however to help integration with content management systems and portals, you can configure WebLogic Mobility Server to allow for ampersands inside attributes. To configure this feature, see the section “Enable Less Strict Document Checking.”

This query string could cause errors:

```
<a href="/blah/checkout.jsp?item=2&qty=70&id=JoeBloggs">This might not work.</
a>
```

This is the same query string using entities:

```
<a href="/blah/checkout.jsp?item=2&amp;qty=70&amp;id=JoeBloggs">Notice the
entities replacing the ‘&’s</a>
```

Another potential source of the problem is improperly quoted attributes in the code such as:

```
<img src="/img/blah.jpg height="20" width="10" alt="Blah"/>
```

Badly nested tags can also raise these types of issues. WebLogic Mobility Server normally handles them and, in most cases, a recover should be possible, however the end result may not be as the author intended.

Question 5

I have created a HTML page and have inserted some mmXHTML (<mm-exclude>) tags to exclude content from a "full browser". When I request the page, it seems that WebLogic Mobility Server has not processed the page at all. In addition, there is no output on the WebLogic Mobility Server Application Server console window.

Solution

The most probable cause of this problem is that WebLogic Mobility Server has failed to recognize your page as an mmXHTML page. WebLogic Mobility Server tries to identify the page by using a document header, such as shown here.

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//MOBILEAWARE// DTD MMXHTML 1.2//EN"
"http://www.mobileaware.com/DTD/mmxhtml_1.2.dtd">
```

Insert the preceding header into your page and try again.

Question 6

How do I set the session timeout time in my mobilized web application, now that there is no longer an entry in the *mis.properties* file to do this?

Solution

In the `<web-app></web-app>` section of your *web.xml* file, place (or modify) the following tag after any `<servlet></ servlet>` tags:

```
<session-config><session-timeout>30</ session-timeout></session-config>
```

This example sets the session timeout at 30 minutes.

The location of this tag in the *web.xml* file is important. A parsing exception will occur if the tag is located too early in the file.

In a typical WebLogic Mobility Server servlet filter deployment, the best place to put this tag is just before the `</web-app>` closing tag.

Question 7

WebLogic Mobility Server is not rewriting my URLs to include session ID information. My application is therefore not tracking sessions from browsers that do not support cookies. How can I include the session ID in my URL’s?

Solution

WebLogic Mobility Server does not rewrite session IDs into URLs in content by default. For author-added links, this can be achieved by calling the container’s encodeURL() method on any URLs that should be rewritten to contain session information. To ensure that WebLogic Mobility Server internally uses this mechanism to encode URLs that it generates (such as "next" links in multi-page WAP content) you will need to set:

"generatedLinks.encodeSessionId" to "true" in the application's *mis.properties* file. Refer to any JSP/servlet authoring guide for more details of encodeURL().

Question 8

All of my WAP clients use cookies, but WebLogic Mobility Server still puts a session ID into "next" links, and so on, on the first page accessed in a session, resulting in some very long links. Can this behavior be disabled?

Solution

Yes. Set the property "generatedLinks.encodeSessionId" to "false" in your application's WebLogic Mobility Server properties. This will stop WebLogic Mobility Server from inserting session IDs in any content it generates.

Question 9

Where has *mis.properties* gone?

Solution

The WebLogic Mobility Server properties are now stored in an application-specific properties file, configured in the web.xml deployment descriptor for your web application. The following example tells WebLogic Mobility Server to use a file called *mis.properties* in the web application's CLASSPATH (for example **WEB-INF/classes**).

This is the default setting.

```
 <init-param>
   <param-name>propertiesname</param-name>
   <param-value>/mis.properties</param-value>
</init-param>
```

Question 10

I have created a WebLogic Mobility Server deployment on my content server. I am making modifications to my *mis.properties* file, but the modifications are not being reflected in the deployed application.

Solution

Ensure that it is the properties file that is referenced in the deployed application web.xml that is being modified.

Question 11

When I start the Diagnostics Console, the message "Unable to connect to server. Please verify the deployment path in the Settings Dialog" displays.

Solution

The console is not pointing to a web application that is configured to support diagnostics. Open the settings dialog and ensure the "Deploy Path" is set to **<webappname>/Diagnostics**. If, for example, the application is in **news** under the web application directory, the deploy path should be set to "**news/Diagnostics**".

Question 12

When I start the Diagnostics Console, the message “Unable to connect to server. Please verify the host/port in the Settings Dialog” displays.

Solution

This can result from either the host/port configuration in the Settings Dialog being configured incorrectly or the web server/application server being unavailable. Correct this by changing the host/port combination, by starting up the server or by deploying the web application.