BEA WebLogic Mobility Server

Administration Guide

Version 3.5
March 2007

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

Introduction

About this Manual

This manual introduces you to the various tasks required to configure and manage BEA WebLogic Mobility Server™ version 3.5.

Intended Audience

It is recommended that your IT department, development team or a technical consultant perform the tasks outlined in this document.

Begin by reading the next section, which explains how to use the guide.

Using this Guide

The manual is divided into the five sections—chapters 1-5 explain how to set up and configure WebLogic Mobility Server, describe the various files used by WebLogic Mobility Server and outline the changes that you need to make during configuration.

The “Appendix” chapter provides a sample web application descriptor.

Notes

- This is the third guide that you will use in the process of installing and running the product. Ensure that you have performed the tasks outlined in the *BEA WebLogic Mobility Server Installation Guide* and installed the Device Repository as per the *Device Repository Guide* before proceeding here
- As outlined in the *BEA WebLogic Mobility Server Installation Guide*, ensure that you have also installed the BEA WebLogic Mobility Server license before proceeding
- The term “Mobility Extension for BEA Workshop” used in the document refers to both the Mobility Extension for BEA WebLogic Workshop 8.1 and the Mobility Plugin for BEA Workshop for WebLogic Platform 9.X.
- The term **<WLMS_install_directory>** refers to either **<BEA_install_directory>\weblogic81\mobility** or **<BEA_install_directory>\weblogic92\mobility** depending on your installation
- The directory **<bea>\weblogic81** is used in this document—if you have a BEA WebLogic 9.X installation, use **<bea>\weblogic92** instead

Further Reading

For further information on WebLogic Mobility Server, see the following user guides:

- *BEA WebLogic Mobility Server Installation Guide*
- *Device Repository Guid*e
- *BEA WebLogic Mobility Server Getting Started Guide*
- *BEA WebLogic Mobility Server User Guide*
- *BEA Mobilize Your Portal Guide*
- *BEA Sample Mobility Portal Guide*
- *BEA Sample Workshop Mobility Project Guide*

1—Deploy an Application

In order to develop Java Server Pages or Java Servlets, regardless of whether they are mobilized or not, you must establish a server-side environment that interprets, compiles and executes the pages that you write.

This section describes how to deploy a mobilized web application with the WebLogic Mobility Server servlet filter.

There are two key steps to deploying the WebLogic Mobility Server applications in a servlet environment:

- Creating the deployment environment
- Configuring the web deployment descriptor

Create the Deployment Site

Servlet containers such as BEA WebLogic support the Java Servlet 2.3 specification that standardizes how the various components in a web application are organized. Some things to remember:

- You store all the files for a web application under a root directory. This root directory also serves as the document root for this web application. Usually, you would place the root directory within the applications directory of your container
- You store all application files (such as the JSP pages, HTML pages and images) under the root directory
- A special directory named **WEB-INF** contains the information describing the web application and how it should be configured and how it should behave
- For a new WebLogic Mobility Server deployment, copy the contents of **<WLMS_install_directory>\lib** directory to **WEB-INF\lib**

Note: If you developed your mobilized application in BEA WebLogic Workshop with WebLogic Mobility Server installed, “Enable Multi-Channel” has already copied any necessary WebLogic Mobility Server files into the WEB-INF directory.

The following table shows the contents of the **WEB-INF** directory:

WEB-INF Directory Contents

WEB-INF Contents	Description
/WEB-INF/web.xml	The deployment descriptor for the application.
/WEB-INF/classes/*	A directory for Java utility classes and other files such as the *mis.properties* and *oscache.properties* files. For further details on how to set up this directory, see the section “Configure the Web Deployment Descriptor”.
/WEB-INF/lib/*.jar	A directory for Java ARchive (JAR) files that contain servlets, JavaBeans, and other utility classes for the web application. The container automatically loads these classes. Copy the contents of the **<WLMS_install_directory>\lib** to this folder.

For additional information on deploying applications on BEA WebLogic Platform, please see the Developing Web Applications for BEA WebLogic Server documentation:

http://e-docs.bea.com/wls/docs81/webapp/index.html

Configure the Web Deployment Descriptor

Before deploying a web application, you need to place a deployment descriptor (*web.xml*) into the **WEB-INF** directory. This file pulls together all the components of the Web application.

Note: For a sample *web.xml* file, see the "Appendix".

The following table describes the key elements in the *web.xml* file.

Key Elements in web.xml File

Element	Description
<web-app>	This is the root element for the deployment descriptor.
<display-name>	Specifies the name to be displayed for the application.
<filter>	Names the filter used by this web application and the parameters it receives.
<filter-mapping>	Specifies which URL pattern is mapped to the servlet.

Configure the WebLogic Mobility Server Filter and Filter Mapping

An example of a *web.xml* file that has been configured for the WebLogic Mobility Server filter is shown here followed by an explanation of the settings.

```
<filter>
    <filter-name>mobilityFilter</filter-name>
    <display-name>Mobility Filter</display-name>
    <description>Mobility Filter</description>
    <filter-class>com.mobileaware.mcp.MobilityFilter</filter-class>
    <init-param>
        <param-name>propertiesname</param-name>
        <param-value>/mis.properties</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>mobilityFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

WebLogic Mobility Server web.xml Filter Settings

Setting	Description
<filter-name>	The name of the filter used by this web application: mobilityFilter
<display-name>	The name of the filter to be displayed: WebLogic Mobility Server Mobility Filter
<description>	A description of the filter: WebLogic Mobility Server Mobility Filter
<filter-class>	The class file containing the filter: com.mobileaware.mcp.MobilityFilter
<param-name>	A name for each parameter to be passed to WebLogic Mobility Server: propertiesname
<param-value>	A value for each corresponding parameter name. In this case, the name and location of the properties file. /mis.properties

WebLogic Mobility Server web.xml Filter Mapping Settings

Setting	Description
<filter-name>	Specifies which URL pattern is to be mapped for the servlet specified by <filter-name>: mobilityFilter
<url-pattern>	The URL pattern to follow:/*

For more details on the Java Server specification and how to configure the *web.xml* file, please use the following URLs:

- Java Servlet Specification:
  http://www.jcp.org/aboutJava/communityprocess/final/jsr053/
- General Java servlet technology documents:
  http://java.sun.com/products/servlet/docs.html
- Developing Web Applications for BEA WebLogic Server documentation:
  http://e-docs.bea.com/wls/docs81/webapp/index.html

Deploy a WebLogic Mobility Server Application as a Web Archive File

In many application servers it is standard practice to deploy web applications as web archive (WAR) files. To deploy a WebLogic Mobility Server web application as a WAR file:

1. Using a command line window, navigate to the root directory of your WebLogic Mobility Server web application, for example, **webapps/news**.
2. Use the "jar" utility from the Java[tm] Development Kit distribution to create the WAR file. For example, to create a WAR file for a news application, you would run the following command:

   ```
   jar -cvf news.war *.*
   ```

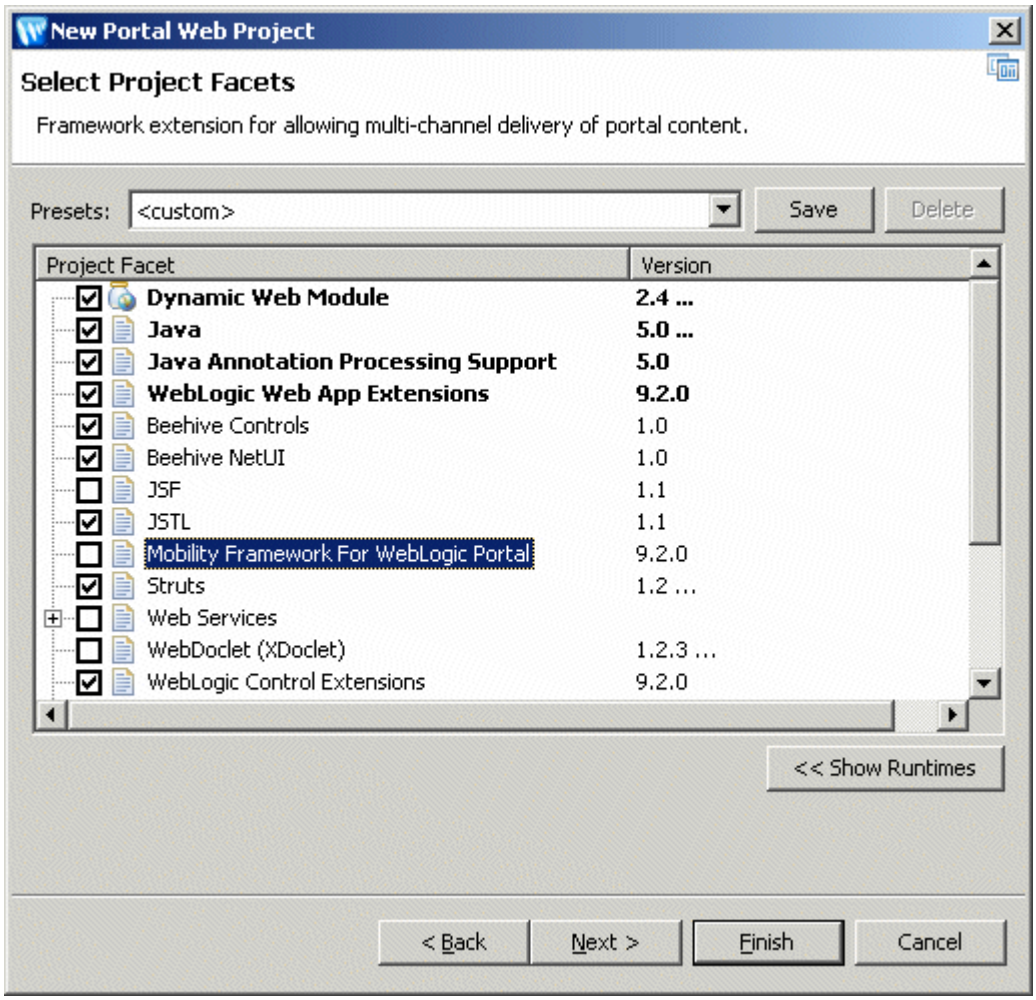
Deploying WebLogic Mobility Server as an EAR file in BEA WebLogic 9.2 Portal

If you select the Mobility Framework for WebLogic Portal facet when creating an EAR file and attempt to deploy this file on another WebLogic 9.2 instance, you will receive an error unless you perform the following step:

Necessary step: Deploy the **portal/lib/modules/***wlp-mobility-web-lib.war* file as a library onto the new server using the WLS Admin Console. This deployment adds an entry to the server's *config.xml* file and resolves the issue.

Mobility Framework for WebLogic Portal 9.2

The Select Project Facets panel is displayed when you create new BEA WebLogic Portal portal web projects, dynamic web projects or EAR projects. Select the **Mobility Framework for WebLogic Portal** to insert the Mobility Framework into your portal project….



Note: You can add facets when creating the projects, or you can return and add them in at a later stage.

2—Configure BEA WebLogic Mobility Server

Configure the WebLogic Mobility Server Filter

Configuring a WebLogic Mobility Server filter for each web application involves creating a *mis.properties* file that specifies:

- the database that WebLogic Mobility Server needs to connect with
- the error pages to be returned in the event of any HTTP- or WAP-generated errors
- a number of additional configuration settings as described in "Configure BEA WebLogic Mobility Server"

There is a sample properties file, *mis.properties.sample*, in the **lib** folder of the installation directory. Copy this file into **WEB-INF/classes** folder and make the necessary changes for configuring the database and the error pages.

You can rename the file to a name of your own choosing, but you need to specify the name in <param-value> section of the *web.xml* file that has been configured for the filter. For further details, see the section "Configure the WebLogic Mobility Server Filter and Filter Mapping".

This section introduces the various properties used to manage the behavior of WebLogic Mobility Server. As an administrator, you will need to configure some of these properties to adjust the behavior of WebLogic Mobility Server.

Some of these properties will have been set during the install process, while others can be configured later when you want to activate other new features.

Configure the mis.properties File

The main configuration file used by WebLogic Mobility Server, which contains a number of user-modifiable parameters, is the *mis.properties* file. This file can be found in the **WEB-INF/classes** folder. It is a plain text file that can be edited in any text editor.

This section describes configuration settings for the *mis.properties* file.

Configure the mis.properties File for the Device Repository

The Device Repository can be deployed as either a database or a *DeviceRepository* file. If it is deployed as a database, follow the instructions in the "Configure a database Device Repository" section; if it is deployed as a file, follow the instructions in the "Configure a File-Based Device Repository" section.

Note: This information is also documented in chapter 2 of the *Device Repository Guide*

Configure a Database Device Repository

The following Device Repository properties must be configured in order for WebLogic Mobility Server to successfully communicate with the Device Repository used:

Device Repository Properties Settings

Property	Description
deviceDB.driver	Location of the JDBC driver to be used by WebLogic Mobility Server to gain access to the database. This property also has the effect of informing WebLogic Mobility Server which database it is connected to. For Oracle, set to: oracle.jdbc.driver.OracleDriver For MySQL, set to: org.gjt.mm.mysql.Driver For Postgres, set to: org.postgresql.Driver For PointBase, set to: com.pointbase.jdbc.jdbcUniversalDriver For SQL Server (with WebLogic Mobility Server deployed on BEA WebLogic only), set to: weblogic.jdbc.sqlserver.SQLServerDriver For Sybase ASE set to: com.sybase.jdbc2.jdbc.SybDriver For IBM DB2 Universal Database set to: com.ibm.db2.jcc.DB2Driver To configure WebLogic Mobility Server to use the BEA WebLogic database connection pool: weblogic.jdbc.pool.Driver Example: deviceDB.driver:oracle.jdbc.driver.OracleDriver
deviceDB.url	The URL to use to access the Device Repository. For Oracle, set to: jdbc:oracle:thin:@<oracle_host>:<oracle_port>:<oracle_database_name> For MySQL, set to: jdbc:mysql://<mysql-server-ip:port>/ <db-name>?user=<connect-user>&password=<connect-password> For Postgres, set to:

	jdbc:postgresql://<postgres_machine>:<postgres_port>/<postgres_database_name> For PointBase, set to: jdbc:pointbase:server://<pointbase_machine>:<pointbase_port>/cajun For SQLServer (with WebLogic Mobility Server deployed on BEA WebLogic only), set to: jdbc:bea:sqlserver://<sqlserver_host>:<sqlserver_port>;databaseName=<sqlserver_database_name> For Sybase ASE set to: jdbc:sybase:Tds:<ip_address>:<port>/SID For IBM DB2 Universal Database set to: jdbc:db2://<ip_address>:<port>/SID When using WebLogic database connection pool, set to: jdbc:weblogic:pool:<poolname> Example: deviceDB.url: jdbc:oracle:thin:@oracle_host:1521:mySID
deviceDB.user	Username used by WebLogic Mobility Server to access the database server. Note: For MySQL, this property is left blank. Example: deviceDB.user: user
deviceDB.password	Password of user used by WebLogic Mobility Server to access the database server. For MySQL, this property is left blank. Example: deviceDB.password: password
deviceDB.maxDBConnections	A numeric value indicating the number of concurrent database connections in the database pool. This is used to control the number of concurrent database connections and licenses required by WebLogic Mobility Server. Defaults to 10. See the section “About Connection Pools” for more information. Example: deviceDB.maxDBConnections: 10
deviceDB.waitTime	A numeric value indicating (in milliseconds) the time to wait for a connection from the database pool. Defaults to 5000. Example: deviceDB.waitTime: 5000
deviceDB.increment	A numeric value indicating the number of connections to add to the pool if there are no connections currently available. If the maximum number of connections in the pool has been reached then no new connections will be added to the pool. Defaults to 1 Example: deviceDB.increment: 1

About Connection Pools

A dynamic web site often generates HTML pages from information stored in a database. Each request for a page results in a database access. Connecting to a database is time consuming since the database must allocate communication and memory resources as well as authenticates the user and set up the corresponding security context. Setting up the individual connections can become a bottleneck.

Establishing the connection once and using the same connection for subsequent requests can therefore dramatically improve the performance of a database driven web application. Connection

pooling is a technique used to avoid the overhead of making a new database connection every time an application or server object requires access to a database. Rather than making and breaking connections as required, a "pool" of database connections is maintained by the system on the server. When WebLogic Mobility Server needs a database connection, it simply requests an available one from the pool - if none is available, a new one is created & added to the pool.

The connection pool not only grows to specified limits, but also contracts as required, closing connections that have not been used for a specified time. This avoids taking up system resources by simply holding connections that are not currently required. This also handles databases which "time-out" their connections, and prevents handing a "stale" connection to an application object.

Configure a File-Based Device Repository

To configure WebLogic Mobility Server to use a file-based Device Repository (i.e. "*DeviceRepository.xml*" or "*DeviceRepository.madr*") instead of connecting to an external database (for example, Oracle, MySQL) where the Device Repository has been installed, you must properly define the database settings in the *mis.properties* file associated with the web applications.

Note: Files with ".madr" extensions contain compressed xml device repositories—see the *Device Repository Guide* for more information on how the Device Repository is created.

You may deploy the Device Repository file in one of two ways:

- In an absolute location
- On the CLASSPATH

Deploy the Device Repository File in an Absolute Location

Locate the *mis.properties* file for your web application in the appropriate **WEB-INF\classes** directory and open it in a text editor.

1. Look for the Device Repository Type setting in the *mis.properties* file, similar to :

```
#########################################################
#
# Device Repository Type
# ---------------------------------------
# This setting indicates whether the Device Repository is
# deployed as a file or installed into a JDBC database.
# Possible values are: xml and db
#
# If not specified, db is assumed.
#
# Note: "xml" is used for both ".xml" and ".madr" Device Repository files
#
deviceRepositoryType: db
```

   Uncomment the last line so that it now reads:

```
deviceRepositoryType: xml
```

2. Look for the Device Repository File Location setting in the *mis.properties* file, similar to:

```
#########################################################
#
# Device Repository File Location
# ---------------------------------------
# This setting indicates the location of the Device Repository file
# This setting must be set to the location of the file or the
# classpath will be checked (see deviceXML.resourceName)
#
# Example:
#C:/bea81sp3/weblogic81/mobility/database/DeviceRepository.madr
#
#deviceXML.location:
C:/bea81sp3/weblogic81/mobility/database/DeviceRepository.madr
```

   Uncomment the “deviceXML.location:” line and change the indicated location to the actual location of the *DeviceRepository* file. The *DeviceRepository* file included with the product is located at: **<WLMS_install_directory>/database/**.

For example, **C/:/bea/weblogic81/mobility/database/***DeviceRepository.madr*.

3. Save the *mis.properties* file.
4. In a BEA WebLogic production environment, you must now use the BEA WebLogic Administration Console to redeploy your web application. In a development environment, the web application can simply be redeployed directly from within BEA WebLogic Workshop.

Deploy the Device Repository file on the CLASSPATH

1. Either add the directory containing the Device Repository file to the CLASSPATH, or deploy the Device Repository file onto either the system or application CLASSPATH.
2. Locate the *mis.properties* file for your web application and open it in a text editor (for example, a sample news application may be located at: **<WLMS_install_directory>\samples\news\WEB-INF\classes**).
3. Look for the Device Repository Type setting in the *mis.properties* file, similar to :

```
#########################################################
#
# Device Repository Type
# ---------------------------------------
# This setting indicates whether the Device Repository is
# deployed as a file or installed into a JDBC database.
# Possible values are: xml and db
#
# If not specified, db is assumed.
#
# Note: "xml" is used for both ".xml" and ".madr" Device Repository files
#
deviceRepositoryType: db
```

4. Change the last line so that it now reads:

```
deviceRepositoryType: xml
```

5. Look for the Device Repository File ResourceName setting in the *mis.properties* file; e.g.:

```
#########################################################
#
# Device Repository File ResourceName
# ---------------------------------------
# This setting indicates the name of the Device Repository file
# when it is deployed as a resource on the classpath.
#
# Example:
# /DeviceRepository.madr
#
#deviceXML.ResourceName: /DeviceRepository.madr
```

6. Uncomment the ‘deviceXML.resourceName:’ line and change the filename if necessary.

   Note: It is important that you do not remove the “/” from the beginning of the line.
7. Save the *mis.properties* file.
8. In a production environment, you must now re-deploy or re-start your web applications.

Configure Session Encoding of URLs (Configuration Optional)

Where session cookies are not supported by devices or gateways, session information can be automatically encoded into URLs using the settings in the table.

Session Encoding of URLs Properties Settings

Property	Description
generatedLinks.encodeSessionId	For URLs generated by WebLogic Mobility Server, this property defines whether WebLogic Mobility Server calls the application server's encodeURL() method to automatically append a session ID. By default, this is set to true. When this property is set to false it will stop WebLogic Mobility Server from inserting session IDs in any content it generates. If set to false, all devices or gateways which connect to WebLogic Mobility Server must support session cookies. Example: generatedLinks.encodeSessionId: false
rewriteAllUrls	This property defines whether WebLogic Mobility Server calls the application server's encodeURL() method to automatically append a session ID for URLs not generated by WebLogic Mobility Server. By default, this is set to false. If set to false, either all such URLs must be manually encoded with the session ID or all devices or gateways which connect to WebLogic Mobility Server must support session cookies. Example: rewriteAllUrls: true

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
FeaturesList.xls&com.bea.event.documenttype=AvitekDocs&contentId=Avitek/DemoDoc
uments/Demo 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 table shows sample URL Compression configuration for WebLogic Mobility Server running against a BEA WebLogic Portal server.

URL Compression Properties Settings for WebLogic Mobility Server Running Against 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 which 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 that have values 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.fail.redirect	The URL to which WebLogic Mobility Server will redirect a request if unrecognized compression tokens are received. This can happen, for example, if the client's session has expired and a page is refreshed or a bookmark is visited. A sensible value for this property would be the home page or login page of the site.

	Note: the URL specified is webapp relative) Example from BEA WebLogic Portal deployment: url.compression.fail.redirect: /avitekfinancial/application/

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.

Error Handling (Configuration Optional)

The location of Error handler JSPs for HTTP and WAP can be configured using the parameters in the table. The location of the JSP error handlers is a webapp relative path.

JSP Error Page Properties Settings

Property	Description
error.handler.jsp	Location of the HTTP JSP error handler, or your own custom file. Example: /errorHandler.jsp
error.handler.wap.jsp	Location of the WAP JSP error handler, or your own custom file. Example: /errorHandlerWap.jsp
errorhandlerXhtml.jsp	Location of the XHTML JSP error handler, or your own custom file. Example: /errorHandlerXhtml.jsp

Sample JSP error-handling pages are provided at **<WLMS_install_directory>\samples\BEAWorkshop\restaurantWeb_After**. These can be copied into your own web-app and customized appropriately.

For more information on configuring error-handling pages, see chapter 5, “Configure the Error Pages”.

Configuration Mode (Configuration Optional)

This configuration entry determines which mode of operation WebLogic Mobility Server runs in.

Configuration Mode Properties Setting

Property	Description
operation.mode	WebLogic Mobility Server has two modes of operation: 'development' and 'production'. Setting the mode to 'development' provides detailed informative warning messages to enable content developers to tune and troubleshoot content during the development phase. By default, operation mode is set to 'production'. Example: operation.mode: development

Generated URLs

WebLogic Mobility Server automatically generates a number of URLs during content transformation. For content that WebLogic Mobility Server produces for WAP phones, WebLogic Mobility Server can produce 'Next' and 'Back to Top' links that contain the identifier of the required page. Similarly, when WebLogic Mobility Server splits a form into multiple pages, the URLs generated by WebLogic Mobility Server contain information about required form ID, current form ID and whether a form reset has been requested.

The parameter name that WebLogic Mobility Server uses for these identifiers can be changed in the *mis.properties* file, in the event that they clash with those already used by content developers.

URL Generation Properties Settings

Property	Description
form.currpagenumber.paramname	In delivering forms to menu-driven devices, WebLogic Mobility Server splits large documents into numbered forms. WebLogic Mobility Server uses the value of this property to create any URLs that reference the current page in a paginated form. By default, this is set to c_-p.
form.nextpagenumber.paramname	Defines the parameter name used by WebLogic Mobility Server to reference the number of the next page in a paginated form. By default, this is set to form_n_-p.
form.uniqueid.paramname	Defines the parameter name for WebLogic Mobility Server to use in generated URLs which reference the session-wide form identifier. By default, this is set to form_-id.
url.idomid.paramname	Defines the name of the parameter that WebLogic Mobility Server uses to uniquely reference parsed documents in generating 'back to top' links.

	By default, this is set to page_id.
form.reset.paramname	Defines the name of the parameter that WebLogic Mobility Server places in URLs to indicate that a form-submit is actually a ‘reset’. By default, this is set to form_-reset.
url.pagenumber.paramname	In delivering content for WAP, WebLogic Mobility Server splits large documents into numbered pages and delivers one page at a time. In so doing, WebLogic Mobility Server must add a parameter to certain URLs so they explicitly reference an individual page of transformed content. By default, this is set to n_-p.

Strict Attribute Handling and Delivery (configuration optional)

These configuration entries define whether WebLogic Mobility Server rejects stand-alone ‘&’ symbols, and whether it delivers them in HTML content.

Example of malformed xml:

```
<a href="/url?param1=value1&param2=value">
```

Example of well-formed equivalent:

```
<a href="/url?param1=value1&amp;param2=value">
```

Strict Attribute Handling and Delivery Properties Settings

Property	Description
xsp.strictAttribute	For consistency with XHTML standards, WebLogic Mobility Server parser is configured by default to reject stand-alone ‘&’ symbols in XHTML attributes. For integration with pre-existing content and frameworks, this strictness can be switched off by setting this property to false. Allowable values are “true” and “false”. Example/Default: xsp.strictAttribute: true
html.deliverStrictAttribute	When enabled, urls with query string parameters use the full XML entity reference "&" and thus will be delivered in the form: /<file>?x=1&y=2&z=3 When disabled, these URLs take the form: /<file>?x=1&y=2&z=3 Default: "false" (that is, use '&'). Allowable values are “true” and “false”. Example: html.deliverStrictAttribute: true

Diagnostics Subscriptions (Configuration Optional)

WebLogic Mobility Server is configured by default to send certain important diagnostics messages, such as error messages, to the application server console. It is possible to configure these and additional diagnostic messages to be sent either to the console or to a specified file.

Example Diagnostics Messages Published to File

```
diagnostics.startup.subscriptions.abcFile.topic: MIS.General.Startup
diagnostics.startup.subscriptions.abcFile.level: verbose|normal
diagnostics.startup.subscriptions.abcFile.filename: c:/diagerrors.log
```

Notes

- The term "abc" is simply a placeholder for a unique identifier, to ensure that property names are unique. You are free to choose your own identifier
- Each topic you subscribe to must be configured to output to a different file
- At start-up, the specified file is overwritten, not appended to

Example Diagnostics Messages Published to the Console

```
diagnostics.startup.subscriptions.xyzConsole.topic: MIS.General.Startup
diagnostics.startup.subscriptions.xyzConsole.level: verbose|normal
```

By default, the WebLogic Mobility Server diagnostics are configured to publish start-up messages and diagnostic-audit messages (to track individuals connecting to diagnostics) to the console as follows:

```
diagnostics.startup.subscriptions.startupConsole.topic:MIS.General.Startup
diagnostics.startup.subscriptions.startupConsole.level:normal
diagnostics.startup.subscriptions.diagnosticsauditConsole.topic:MIS.Diagnostics
.Subscription
diagnostics.startup.subscriptions.diagnosticsauditConsole.level:normal
```

Error messages could additionally be configured to publish to file as follows:

```
diagnostics.startup.subscriptions.errorsFile.topic:
diagnostics.startup.subscriptions.errorsFile.level:verbose
diagnostics.startup.subscriptions.errorsFile.filename:c:/temp/diagerrors.log
```

This is a special case where no topic is required.

Diagnostics Subscriptions Properties Settings

Property	Description
diagnostics.startup.subscriptions.xxxFile.topic:	Any diagnostic topic which can be selected from the WebLogic Mobility Server Diagnostics
diagnostics.startup.subscriptions.xxxFile.level:verbose	Specifies the level of diagnostics message required – either “verbose” or “normal”
diagnostics.startup.subscriptions.xxxxFile.filename:	Name of file to log this diagnostic subscription to

Back to Top (Configuration Optional)

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

When 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. If the user then uses this enhancement to navigate to the top-level navigation card of the current document hierarchy, they are provided with a “Back To Top” link that returns them to the referrer.

Back to Top Properties Setting

Property	Description
backtotop.enabled	Indicates whether “Back to Top” links should be used for paginated content. Default: true

Content Length Settings

The `response.omitContentLength` configuration entry defines whether or not WebLogic Mobility Server will set the content length in the response. By default, `response.omitContentLength` is set to false, implying that WebLogic Mobility Server will set the content length in the response. If it is set to true WebLogic Mobility Server will not set the content length and chunked encoding will be used to deliver content.

Content Length Properties Setting

Property	Description
response.omitContentLength	Indicates whether the content length should be included in the response from WebLogic Mobility Server Default: false.

Display “Submit” and “Previous” buttons on Each Page of Form (XHTML-MP Devices Only)

The following *mis.properites* attributes allow the author to add “submit” and “previous” buttons to each page of a form during pagination, if required.

mis.properties attribute	Feature Description
form.subpage.submit	True I false (default) Set to true results in a “submit” button displayed on each page of a form. If the user selects the “submit” button, WLMS will submit all user inputs for the current and preceeding pages of the form. Set to false results in a “submit” button only at end of pagination.
form.subpage.previous	True I false (default) Set to true results in a “previous” button on each page of a form (except first). If the user selects the “previous” button, WLMS will clear all user input for the current and preceeding pages. The user will then be returned to the previous page of the form. Set to false results in a “previous” button not shown.

Edit the button text via the *contentassembly.properties* file—see chapter 3.

Disallowed Output Encodings (Configuration Optional)

WebLogic Mobility Server determines from the incoming device request which character encodings will give the best rendering of content. In some circumstances, however, a device may incorrectly report its quality of support for a given character encoding, or there may be no valid transformation from the original content encoding to the preferred device encoding. Specifying a comma-separated list of encodings for the `disallowedOutputEncodings` property instructs WebLogic Mobility Server never to deliver content in any of these encodings.

Disallowed Output Encodings Properties Setting

Property	Description
disallowedOutputEncodings	Indicates output encodings that WebLogic Mobility Server should never use. Example: disallowedOutputEncodings: iso-8859-1, iso-8859-5

Optimize Performance with the JSP Tag Library (Configuration Optional)

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.

For full details on achieving optimal performance with the JSP Tag Library, see the section "Optimizing Performance with the JSP Tag Library" in the *BEA WebLogic Mobility Server User Guide*. The properties involved in this process are summarized.

Optimizing JSP Tag Library Performance Properties Setting

Property	Description
mis.jsptaglib.passthrough	Optimizes processing when JSP files are known to contain only mm: and non mm- tags. Allowable values are "true" and "false". Example: mis.jsptaglib.passthrough: true
mis.passthrough.patterns	Optimizes processing when JSP files matching specified patterns are known to contain only mm: and non mm- tags. Example: mis.passthrough.patterns: *.jsp
bypass.device.recognition	Bypasses certain processing where JSP files matching specified patterns are to be delivered only to FullBrowsers. Example: mis.bypass.patterns: /pc/*.jsp
mis.fullbrowser.device	Specifies fullbrowser device to be used with mis.bypass.patterns, or in unlicensed mode. Example: mis.fullbrowser.device: Mozilla/5

Passthrough Pattern Configuration (Configuration Optional)

The following properties control which HTTP requests WebLogic Mobility Server will act on. For performance reasons it is beneficial to be able to inform WebLogic Mobility Server to not process any request that will not produce MMXHTML.

When an HTTP request is received, WebLogic Mobility Server will check the URL against the "mis.patterns.url.nonmmxhtml" patterns. If it matches, the request will not be processed.

Otherwise, WebLogic Mobility Server will check the "mis.patterns.url.mmxhtml" patterns. If these match, or the property "mis.patterns.url.unknown.mmxhtml" is set to true, the request is processed and the Content-Type is checked after the content has been produced.

The Content-Type check is done in much the same was as the URL check. The Content-Type is checked against the "mis.patterns.contenttype.nonmmxhtml" property. If it matches, the content is delivered as produced. Otherwise, if either the "mis.patterns.contenttype.unknown.mmxhtml" is set to true or the "mis.patterns.contenttype.mmxhtml" pattern matches the content is transformed.

Note: If you need to modify or add to these lists you need to include the appropriate values from the default settings as you are overriding the property. Only remove from these lists if you are sure that is what you want to do. Configuring these properties incorrectly may cause WebLogic Mobility Server to no longer process content.

In the following properties the "patterns" may be of the form:

XXX* - starts with XXX, for example /images/*
*XXX - ends with XXX, for example *.gif
XXX - contains XXX, for example */ignore/*

Passthrough Pattern Properties Setting

Property	Description
mis.patterns.url.nonmmxhtml	Configure the list of URL patterns to NOT consider MMXHTML. If a request is received for a URL matching one of these patterns it will not be processed by WebLogic Mobility Server. Default: mis.patterns.url.nonmmxhtml: *.css *.gif *.jpg *.jpeg *.jpe *.wbmp *.swf *.dwt *.ico *.png *.txt *.pdf
mis.patterns.url.mmxhtml	Configure the list of URL patterns to consider potentially MMXHTML. If a request is received for a url matching one of these patterns it will be processed by WebLogic Mobility Server. Default: mis.patterns.url.mmxhtml=*.htm *.html *.jsp
mis.patterns.url.unknown.mmxhtml	Configure if an unknown URL should be considered potentially MMXHTML. If a request is received for a URL not matching the "mis.patterns.url.nonmmxhtml" or the "mis.patterns.url.mmxhtml" patterns, this property decides if it should be considered MMXHTML. Default: mis.patterns.url.unknown.mmxhtml: true
mis.patterns.contenttype.nonmmxhtml	Configure the list of Content-Type patterns to NOT consider MMXHTML. If a response is received with a Content-Type matching one of these patterns it will not be processed by WebLogic Mobility Server.

	Default: mis.patterns.contenttype.nonmmxhtml: application/* audio/* image/* message/* model/* multipart/* video/* text/css text/plain text/rtf text/vnd* text/xml
mis.patterns.contenttype.mmxhtml	Configure the list of Content-Type patterns to consider MMXHTML. If a response is received with a Content-Type matching one of these patterns it will be processed by WebLogic Mobility Server. Default: mis.patterns.contenttype.mmxhtml: text/html;* text/html text/tml text/tml;*
mis.patterns.contenttype.unknown.mmxhtml	Configure if an unknown Content-Type should be considered MMXHTML. If a request is received for a Content-Type not matching the "mis.patterns.contenttype.nonmmxhtml" or the "mis.patterns.contenttype.mmxhtml" patterns, this property decides if it should be considered MMXHTML. Default: mis.patterns.contenttype.unknown.mmxhtml: false
mis.patterns.contenttype.askjava	Configure whether content type should be ascertained by asking java to do the mapping from URL to mime-type. Default: mis.patterns.contenttype.askjava:true

Reverse DNS Lookup Settings

The `diagnostics.enableReverseDNS` configuration entry defines whether or not WebLogic Mobility Server enables or disables the remotehost name resolution taking place in the Diagnostic Context for every request. By default, `diagnostics.enableReverseDNS` is set to false, ensuring that WebLogic Mobility Server will disable the remotehost name resolution.

Content Length Properties Setting

Property	Description
diagnostics.enableReverseDNS	Defines whether or not WebLogic Mobility Server enables or disables the remotehost name resolution taking place in the Diagnostic Context for every request. Default: false.

BEA WebLogic Portal Settings (Configuration Mandatory for BEA Portal Deployment)

The following properties are to facilitate WebLogic Mobility Server Integration with BEA WebLogic Portal and Server.

Note: Do not change these settings if running with BEA WebLogic Portal or Server.

BEA WebLogic Portal Properties Setting

Property	Description
compatibility.illegalState.weblogic	Must be set to true if WebLogic Mobility Server deployed with BEA WebLogic Portal or Server. This is set by default when using the “Enable Multi-Channel” option in BEA WebLogic Workshop
bea.portal.integrationOn	Must be set to true if WebLogic Mobility Server deployed with BEA WebLogic Portal or Server. This is set by default when using the “Enable Multi-Channel” option in BEA WebLogic Workshop

Specify Locales

By default, WebLogic Mobility Server is configured to use the default locale as specified in the Accept-Language header on an incoming request. If no locale is specified in the Accept-Language header, WebLogic Mobility Server will use the locale set on the operating system on which it is running. However, it is possible to create a customized class that WebLogic Mobility Server can use to specify a different locale, for example, one preferred by the user. The locale.customClass property, in the *mis.properties* file, needs to be configured with the name of this class. WebLogic Mobility Server will use then this class to determine the required locale for each request.

Note: If the property is not set, does not exist or has an invalid setting, WebLogic Mobility Server will use the default behavior described here.

locale.customClass property setting

Property	Description
locale.customClass	A reference to a custom class that can be used to override the default selection behaviour used by WebLogic Mobility Server when generating content in reply to a user’s request. Call for each request. Example: locale.customClass: com.mobileaware.util.CustomLocaleFinderImpl

To create a customized locale finder, complete the following steps:

- Create a class to obtain the appropriate locale. Ensure that the class implements the com.mobileaware.util.CustomLocaleFinder interface
- The CustomLocaleFinder interface has one method called getLocale(). This method takes in one parameter- the HttpServletRequest object. This can be used to extract information from the request or Session if necessary
- Either place the class file in an appropriate directory within the **WEB-INF/classes** directory or jar the class file and place it in the **WEB-INF/lib** directory

- In the *mis.properties* file, set the *locale.customClass* property to contain the name of this class

Example

The example illustrates how one can use a customized class to determine the locale in the context of an application. A Web application might use the following to allow the user to choose their preferred language:

A JSP directive to set an arbitrary attribute,"UserPreferredLocale" on the Session:

```
<% request.getSession().setAttribute("UserPreferredLocale","es-ES"); %>
```

A complementary customized class used to access the UserPreferredLocale attribute and create an appropriate Locale:

```
package com.mobileaware.util;
import javax.servlet.http.HttpServletRequest;
import java.util.Locale;

public class CustomLocaleFinderImpl implements CustomLocaleFinder{
	public Locale getLocale(HttpServletRequest req){
		String locale = (String)req.getSession().getAttribute("UserPreferredLocale");
		if (locale!=null){
			String language = locale.substring(0,2);
			String country = locale.substring(3,5);
			return new Locale(language,country);
		}else{
			return null;
		}
	}
}
```

Note: For information on localizing those message and content elements that WebLogic Mobility Server automatically inserts into the content, see the following section, “The ContentAssembly.properties File”.

3—The ContentAssembly.properties File

This file contains additional configurable settings that relate to the way content appears on the screen. Most of these settings have to do with the text displayed in the automatically generated links, which aid navigation around sites being delivered to handheld devices. For example, Next, Back to, Previous, and so on.

Unlike the other two properties files discussed in this section, this file is located in the webapp's **WEB-INF/lib/***mmJSPtaglib.jar* file. This file contains the default property values. If you wish to make changes to this file, unzip the jar, make a copy of the properties file, make the changes; then save it to:

WEB-INF/classes/com/mobility/i18n/resources/ContentAssembly.properties

Multiple versions of this file can be created to provide locale specific property values. Which file is used depends on the language and region settings of the requesting device and the availability of a properties file matching those settings. This mechanism uses the Java Internationalization functionality that provides a standard for application designs that are adaptable to various languages and regions without engineering changes.

As an example, if you had customers in France and Germany who would be accessing your website, you would create two versions of the *ContentAssembly.properties* file and name them:

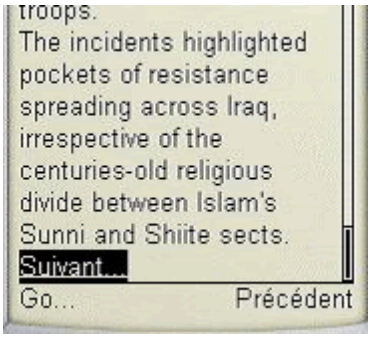
- ContentAssembly_fr.properties
- ContentAssembly.de.properties

Modify the property values to conform to the language of the country.

```
#
# Next Link
#
# The text link that is inserted
#
# during pagination of a group to take user to next
#
# page of content within the group.
#
#
page.group.next.link: Suivant...

#
#
# Previous Link
#
# The text link that is inserted during
#
# pagination of a group to return user to previous page
#
# of the group.
#
############################################################
page.group.previous.link: Précédent
```

Setting properties for French devices



For more information and a list of the fully supported locales and filename extensions, see the Java documentation at java.sun.com.

Content Assembly Properties Settings

Property	Example and Definition
page.group.next.link	More... The text link that is inserted by WebLogic Mobility Server during pagination of a group to take user to next page of content within the group.
page.group.previous.link	Back The text link that is inserted by WebLogic Mobility Server during pagination of a group to return user to previous page of the group.
form.previous.link	Back The button text that is inserted by WebLogic Mobility Server on *each* page of a form during pagination to return the user to previous page of the form.
form.submit.button.default	Submit The default form submit button text when the user does not define it themselves.
document.previous.link	Back to: %previous_page_title%* The text link that is inserted by WebLogic Mobility Server during pagination that would take you to the previous page of content when in a paginated group.
page.group.top.link	Back to: %grp_name% The text link that is inserted by WebLogic Mobility Server during pagination of a group to take the user to the top of the page of content.
table.top.link	Back to:%table_name% The text link that is inserted by WebLogic Mobility Server during pagination of a group to take the user to the top of the page of content.

image.alt.text.default	<blank> Placed in the 'alt' text on an image when no 'alt' text is defined in an mm-img.
table.link.text.default	%table_name% The text link that is used to access the table data.
table.empty.cell.text.default	 The text placed in an empty cell that is NOT a link.
media.object.missing.alt.text	<blank> Placed in the 'alt' text on an image when no media object is defined.
form.next.link	Next The text link included when a form is paginated.
form.reset.button.default	Reset The default form reset button text when the user does not define it themselves
UnNamedLink	Unnamed Link The default link text for the meta tag defining an mm-section.
Options	Options If all the Options links associated with a page do not fit on the transformed page, they will be placed on a separate page. This property specifies the text for the link to the Options page.

4—The oscache.properties File

The *oscache.properties* file is a text file containing the configuration settings that regulate the content caching mechanism. It is located in the **WEB-INF/classes** folder of each webapp running with WebLogic Mobility Server. It is a plain text file and can be modified in any text editor. There is a sample properties file, *oscache.properties.sample,* in the **lib** folder of the installation directory.

Normally, you do not need to modify anything in this file; however, you might want to change the directory that stores the caches. You can reset the directory by changing the *cache.path* property. By default, a cache directory called **tempCache** is created relative to the directory from which WebLogic Mobility Server was launched.

Important Note: If you change the *cache.path* property, you must ensure that the cache has permission to write to the new directory.

5—Configure Error-Handling Pages

WebLogic Mobility Server provides a set of sample error handling pages and images. It is recommended that you replace these with your own error handling pages and images.

Sample Error Handling Pages Provided with WebLogic Mobility Server

Three possible error pages are provided in WebLogic Mobility Server for three different device types (HTML, WML and XHTMLMP).

The error pages are (in order from complex to simple):

1. **JSP pages**

   The JSP error page is used when the error is within the scope of WebLogic Mobility Server. It will present the contents of a JSP to a device when that devices errors.

File	Description
errorhandler.jsp	Defines the jsp that handles HTTP error messages
errorhandlerWap.jsp	Defines the jsp that handles WAP error messages
errorhandlerXhtml.jsp	Defines the jsp that handles XHTML error messages

2. **Sample Simple Error Handling Pages**

   The Simple error page is used when:

   - The error is within the scope of WebLogic Mobility Server
   - There is no defined error JSP or the defined error JSP does not compile, or
   - There an error in accessing the Device Repository

   This error page will present the string defined in the property file to a device when that devices errors.

3. **Sample Default Simple Error Handling Pages**

   The Default error page will be used when:

   - The error is within the scope of WebLogic Mobility Server
   - There is no defined error JSP or the defined error JSP does not compile, and
   - There is no defined simple error page or there is an error in accessing the Device Repository

   This error page will present the string defined in the property file to a device when that devices errors

Locate the Error-Handling Pages

This error pages can be found in the upper directory of the sample BEA Workshop and BEA Portal projects installed during the installation process (see the *BEA WebLogic Mobility Server Installation Guide*).

Copy the error pages to your web application directory or place them in their own directory—the **<webapp>** root—if you prefer.

You will need to modify the following lines in the *mis.properties* file so that they point to the appropriate error handlers.

JSP Error Page Settings

Setting	Description
error.handler.jsp	Location of the HTTP JSP error handler, or your own custom file
error.handler.wap.jsp	Location of the WAP JSP error handler, or your own custom file
error.handler.xhtml.jsp	Location of the XHTML JSP error handler, or your own custom file

Configure the Error-Handling Pages

1. JSP Error-Handling Pages

You can control the JSP sent to the device and the status of the response by changing properties as follows.

Change the JSP Delivered

To change the page returned to you, add or uncomment the following properties within the *mis.properties* file:

```
error.handler.jsp  : location of the HTML JSP handler, or your own custom file
Example: /errorHandler.jsp
error.handler.wap.jsp : location of the WAP JSP handler, or your own custom
file Example: /errorHandlerWap.jsp
error.handler.xhtmlmp.jsp :  location of the XHTMLMP JSP handler, or your own
custom file Example: /errorHandlerXhtmlmp.jsp
```

Change the Status of the Returned Response

There are three error statuses that a JSP error page can have:

1. The status of the failing request

This means that errors will retain their status. This can lead to problems with WML and XHTMLMP device types as they may display no content if the status is that of a serious error. Edit the following settings as required:

```
error.return status set to true
override.status.for.wml.error.response set to false
override.status.for.xhtmlmp.error.response set to false
```

2. The Status of Error JSP

This will return a status of the chosen JSP error page. This allows the writer of the JSP to decide the status given to a device. Edit the following settings as required:

```
error.return status set to false
```

3. A Device-Type Specific Safe status (Default Behavior)

This means that the status of errors will be changed to a value that is safe to display on the requesting device.

- On HTML devices the status will be as the failing request
- On WML devices the status will be 200
- On XHTMLMP devices the status will be 200

Edit the following settings as required:

```
error.return status set to true
override.status.for.wml.error.response set to true
override.status.for.xhtmlmp.error.response set to true
```

2. Simple Error page

You can control the string presented to the device and the status of the response by changing properties as follows.

Change the Simple Error Page Delivered

To change the page returned, add or uncomment the following properties within *mis.properties*:

```
simpleerror.page  : a html page as string
Example: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>simple error page</html>
simpleerror.page.wap : a wml page as string  Example :
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD
WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>simple error page</wml>
simpleerror.page.xhtmlmp :  a xhtmlmp page as string  Example:
<?xml version="1.0" encoding="iso-8859-1"?><!DOCTYPE html PUBLIC "-
//WAPFORUM//DTD XHTML Mobile 1.0//EN"
"http://www.wapforum.org/DTD/xhtml-mobile10.dtd">
<html>simple error page</html>
```

Note: If the string #DESC# is included in the simple error page string it will be substituted with a simple description of the error.

Change the Status of the Returned Response

JSP error pages can have two error statuses:

1. The Status of the Failing Request

This means that errors will retain their status. This can lead to problems with WML and XHTMLMP device types as they may display no content if the status is that of a serious error.

```
override.status.for.wml.error.response set to false
override.status.for.xhtmlmp.error.response set to false
```

2. A device type specific Safe status (Default Behavior)

This means that the status of errors will be changed to a value that is safe to display on the requesting device.

- On HTML devices the status will be as the failing request
- On WML devices the status will be 200
- On XHTMLMP devices the status will be 200

```
override.status.for.wml.error.response set to true
override.status.for.xhtmlmp.error.response set to true
```

3. Default Error page

It is not possible to alter the content of the default error page unless you alter actual code. The various default error pages are as follows:

HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html>
<head>
<title>Error Page</title>
```

```
</head>
<body bgcolor="white">
<b>The following error occurred:</b><br>
#DESC#
<br></body>
</html>
```

WML

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="mm-structure-1" title="Error Page">
<p>
<br/>
The following error occurred :
<br/>
#DESC#
<br/>
</p>
</card>
</wml>
```

XHTMLMP

```
<?xml version=\"1.0\" encoding=\"iso-8859-1\"?>"+
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile 1.0//EN"
"http://www.wapforum.org/DTD/xhtml-mobile10.dtd">
<html>
<head>
<title>Error Page</title>
</head>
<body>
<b>The following error has occurred</b><br/>
#DESC#
<br/></body>"
</html>
```

Change the Status of the Returned Response

This process is identical to that of changing the status of simple error page.

Next Step—Administer the Device Repository

When administering the Device Repository at a later stage, please see chapter 3, “Administer the Device Repository” of the *Device Repository Guide*, which describes how to set up and manage the devices and device profiles stored in the repository.

Appendix—web.xml Sample File

The *web.xml* file is found in the **/WEB-INF** directory of each web application folder. we

<?xml version="1.0" encoding="ISO-8859-1"?> <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd"> <web-app>	Standard web deployment descriptor.
<filter> <filter-name>mobilityFilter</filter-name> <display-name>System Test Filter</display-name> <description>System Test Filter</description>	User-configured filter-name, display-name and description for the instance of the filter.
<filter-class>com.mobileaware.mcp.MobilityFilter</filter-class>	The filter-class must be configured as shown.
<init-param> <param-name>propertiesname</param-name> <param-value>/test.properties</param-value> </init-param>	(Optional) propertiesname. Defaults to "/mis.properties". Defines where WebLogic Mobility Server will look for properties in the CLASSPATH of the webapp.
<init-param> <param-name>namespace</param-name> <param-value>com.mobileaware.mcp</param-value> </init-param> </filter>	(Optional) namespace. Defaults to "com.mobileaware.mcp". This is pre-pended to all session variables and servlet context variables used by WebLogic Mobility Server in the webapp.
<filter-mapping> <filter-name>mobilityFilter</filter-name> <url-pattern>/*</url-pattern> </filter-mapping>	Name of filter as specified above. Path of request to be handled by the filter.
<servlet> <servlet-name>DiagnosticsServlet</servlet-name> <servlet-class> com.mobileaware.diagnostics.http.server.DiagnosticsServlet </servlet-class> </servlet>	Configuration for diagnostic servlet.
<servlet>	Configuration for the

<servlet-name>GUIRequestHandler</servlet-name> <servlet-class> com.mobileaware.MIS.GUI.GUIRequestHandler </servlet-class> </servlet>	Administration Console servlet.
<servlet-mapping> <servlet-name>DiagnosticsServlet</servlet-name> <url-pattern>/Diagnostics/*</url-pattern> </servlet-mapping>	Diagnostics mapping, to be used from TextUI and DiagnosticsConsole.
<servlet-mapping> <servlet-name>DiagnosticsServlet</servlet-name> <url-pattern>/private/Diagnostics/*</url-pattern> </servlet-mapping>	Alternative version of Diagnostics mapping, protected by Application Server security.
<servlet-mapping> <servlet-name>GUIRequestHandler</servlet-name> <url-pattern>/GUIRequestHandler/*</url-pattern> </servlet-mapping>	Administration Console mapping. To be used from the Administration Console.
<security-constraint> <web-resource-collection> <web-resource-name>SecureURLS</web-resource-name> <description>Private</description> <url-pattern>/private/*</url-pattern> <http-method>POST</http-method> <http-method>GET</http-method> </web-resource-collection> <auth-constraint> <description></description> <role-name>Acme</role-name> </auth-constraint> <user-data-constraint> <description>SSL not required</description> <transport-guarantee>NONE</transport-guarantee> </user-data-constraint> </security-constraint> <login-config>	Example of a Security constraint used with Diagnostics

<auth-method>BASIC</auth-method> </login-config> </web-app>	