BEA WebLogic Mobility Server

Device Repository Guide

Version 3.6
June 2007

Copyright

Copyright © 1995-2007 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software is protected by copyright, and may be protected by patent laws. No copying or other use of this software is permitted unless you have entered into a license agreement with BEA authorizing such use. This document is protected by copyright and may not be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form, in whole or in part, without prior consent, in writing, from BEA Systems, Inc.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE DOCUMENTATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA SYSTEMS DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE DOCUMENT IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks and Service Marks

Contents

Introduction

About this Guide

This guide explains how to install the Device Repository used by BEA 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.

Intended Audience

It is recommended that your IT department, database/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 three main chapters:

1. "Install the Device Repository"
2. "Configure the mis.properties Settings".
3. "Administer the Device Repository".

The "Appendixes" chapter provides information on mobile device attributes and the Admin Console tool.

Note: The directory **weblogic81** is used in this document—if you have a BEA WebLogic 9.X installation, use **weblogic92** instead.

Notes

- This is the second guide that you will use in the process of installing and running the WebLogic Mobility Server product. Ensure that you have performed the tasks outlined in the *BEA WebLogic Mobility Server Installation Guide* before proceeding here.

  Once you have installed the product and the Device Repository, you should proceed to the *BEA WebLogic Mobility Server Administration Guide.*

  Chapter 3, "Administer the Device Repository", of this *Device Repository Guide* will then become useful as a reference manual when administering the device profiles

- As outlined in the afore-mentioned guides, ensure that you have also installed the appropriate product 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> denotes either **<BEA_install_directory>**\weblogic81\mobility or **<BEA_install_directory>**\weblogic92\mobility depending on your installation

1—Install the Device Repository

Introduction

WebLogic Mobility Servers require the Device Repository to store device profile information. Follow the instructions and sample screenshots in this chapter to install the Device Repository.

The Device Repository can be deployed as either a *DeviceRepository* file or a database:

- The Device Repository is deployed (as a *DeviceRepository* file) as part of the WebLogic Mobility Server 3.6 install

- If you wish to deploy the Device Repository into a database, you will also need to run the Device Repository Manager tool

The Device Repository currently supports Oracle, MySQL, Postgres, PointBase, SQL Server 2000, IBM Universal DB2 and Sybase Adaptive Server Enterprise databases. The following installation procedures assume that a supported database/the *DeviceRepository* file has already been installed, and that the administrator performing the installation is familiar with database creation.

After completing the Device Repository installation, configure the *mis.properties* settings as described in chapter 2, "Configure the mis.properties Settings".

Perform Pre-Installation Task—Modify Database/Memory Settings

Note: You ONLY need to perform the tasks outlined here if you are installing the full Device Repository into the evaluation PointBase database included in the BEA Portal Domains.

Before you install the Device Repository, complete the steps in either the "Increase the PointBase Settings" section.

Increase the Default Pointbase Settings

You will need to modify the default Pointbase settings if you are installing the full Device Repository into the evaluation Pointbase database included in the BEA Portal Domains.

Open the *pointbase.ini* file from **\bea\user_projects\domains\mydomain** and set values for the following parameters as shown below:

- database.pagesize=10000

- cache.size=10000

- sort.size=10000

The Device Repository Manager Tool

Device Repository Manager is a GUI tool that has three main functions:

- As previously mentioned, the Device Repository can be deployed as either a database or a *DeviceRepository* file. It is deployed as a *DeviceRepository* file as part of the WebLogic Mobility Server 3.6 install. Run the Device Repository Manager too to deploy the Device Repository into a database

- The Device Repository Manager tool then performs subsequent Device Repository updates

- It is also used to perform maintenance on the *DeviceRepository* file-based Device Repository.

Important notes

- The *DeviceRepository* file can be stored and accessed as either an XML file, or in compressed format with the extension ".madr"

   Note: The large XML format device repository file may cause problems when a project is opened in BEA WebLogic Workshop. In this scenario, 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.

- When the Device Repository is represented as a database, you will use the Admin Console tool to add, remove and modify devices and device attributes; for more information, see "Appendix C"

The Device Repository Manager tool itself allows customers to:

- Create a new Device Repository from a flat *DeviceRepository* file (provided by the Online Update Service)

- Backup a customer's existing database to a *DeviceRepository* file

- Access the Device Repository Online Update Service to download and install the latest update provided (also provided as a flat *DeviceRepository* file)

- Add/remove custom devices from the *DeviceRepository* file. This is mainly for use with the *DeviceRepository* file-based repository

- Perform limited modifications on existing device attributes in the *DeviceRepository* file

- Add and remove custom attributes to the *DeviceRepository* file

When using Device Repository Manager to install an update provided by the Online Update Service, the tool:

- Backs up the customer's existing Device Repository to a *DeviceRepository* file

- Detects and stores customer modifications to their existing Device Repository

- Installs the new Device Repository provided by the Online Update Service.

- Presents the customer with a list of modifications and allows the customer to re-apply each of them or accept the values provided in the Device Repository update

Locate the Device Repository Manager Tool

The Device Repository Manager tool can be found under the installation directory that was selected when installing the product:

- On a MS Windows operating system, this would be:
   <WLMS_install_directory>\applications*DeviceRepositoryManager.exe*

- On a UNIX operating system, this would be:
  < WLMS_install_directory>/applications/*DeviceRepositoryManager*

Notes

- Device Repository Manager is a GUI based application that must be run on a system with a windowing environment. You may therefore run it in a UNIX/Linux environment running X Windows, or in a MS Windows environment

- Device Repository Manager connects directly to the database within which the Device Repository is to be installed, so it is not necessary to run it on the same platform on which WebLogic Mobility Server was installed

Pre-Configuration for Support of IBM Universal DB2

If using Device Repository Manager to install the Device Repository on IBM Universal DB 2, copy the following driver files from **<ibm DB2 install_directory>\SQLLIB\java**, for example, **C:\Program Files\IBM\SQLLIB\java** to **<install_directory>\applications\lib**:

- db2jcc.jar

- db2jcc_license_cu.jar

This will enable Device Repository Manager to install and/or update the Device Repository into a configured IBM Universal DB2 database.

Pre-Configuration for Support of SQL Server 2000

If using Device Repository Manager to install the Device Repository on SQL Server 2000, download the necessary JDBC drivers from: **http://www.microsoft.com/downloads/details.aspx?FamilyID=86212d54-8488-481d-b46b-af29bb18e1e5&displaylang=en** and then copy the following files to either **< WLMS_install_directory>\applications\lib**:

- msbase.jar

- mssqlserver.jar

- msutil.jar

This will enable Device Repository Manager to install and/or update the Device Repository into a configured Microsoft SQL Server 2000 database.

Device Repository Manager Scenario 1: Install a New Device Repository

Create a database for the Device Repository and note the connection details. To complete the Device Repository installation, you will need to know the database type, the database URL, and a valid username and password for accessing the database.

1. Run DeviceRepositoryManager.exe (Windows) or DeviceRepositoryManager (UNIX/Linux platforms).

2. The "Device Repository Options" dialog is displayed.

    Device Repository Options dialog

3. Select the **Install/Update Device Repository from File** option to install or update the Device Repository using the *DeviceRepository* file.

4. The "Select Latest Device Repository file" dialog is displayed.

Select Latest Device Repository File dialog
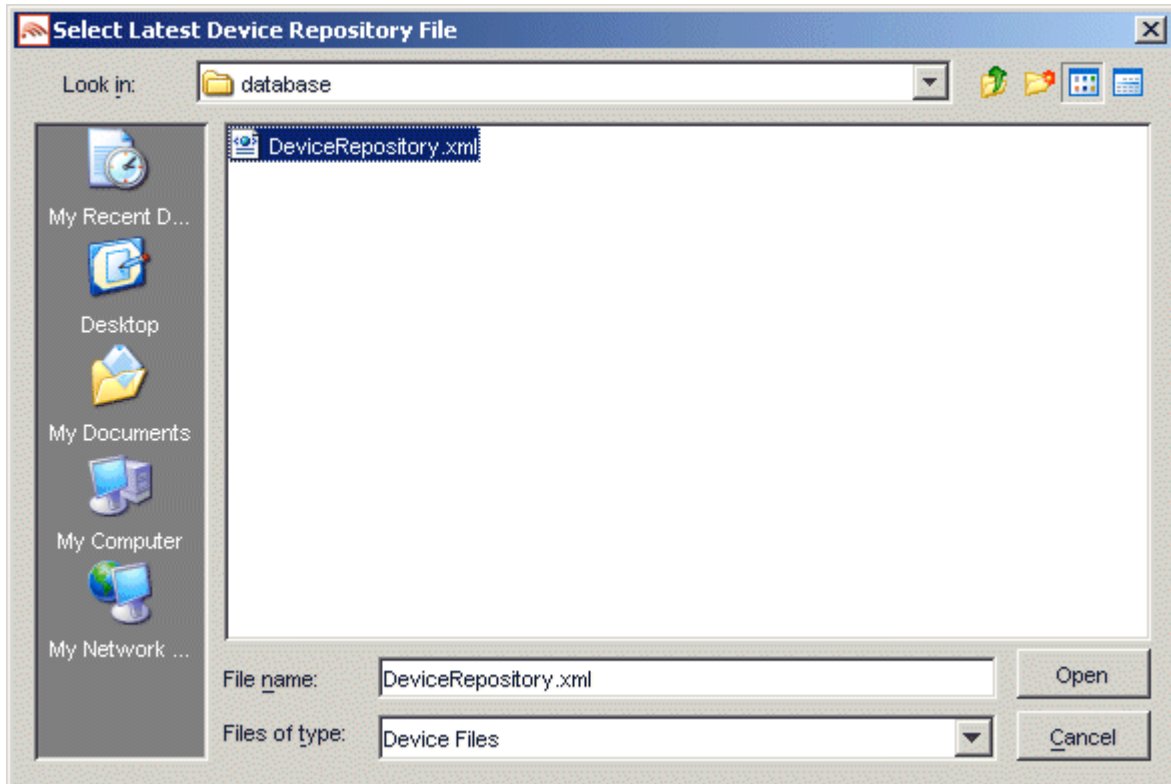
5. Select a *DeviceRepository* file to install and click **Open**. The *DeviceRepository* file included with the WebLogic Mobility Server installer will be shown as the default for a new installation.

6. The "Device Browser" screen is displayed.

   Device Browser screen
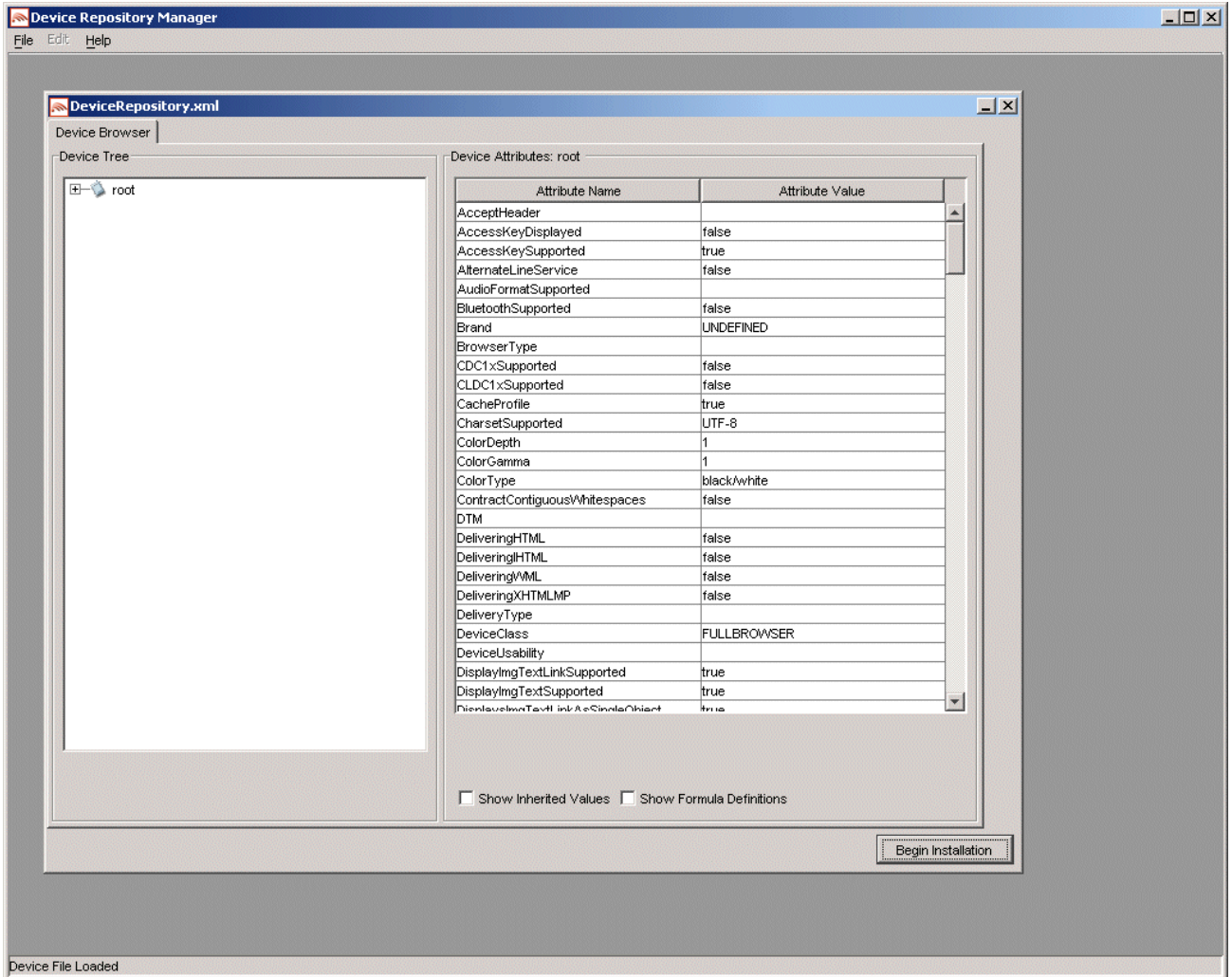
7. This screen displays the device data contained in the *DeviceRepository* file. The data is shown in a hierarchical structure as a preview of the Device Repository to be installed. To display inherited values for each device, select the **Show Inherited Values** check box. Click **Begin Installation** to proceed.

8. The second "Device Repository Options" dialog is displayed.

   Second Device Repository Options dialog
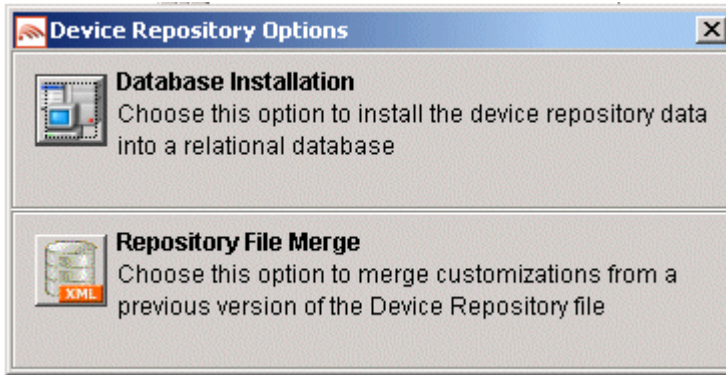
9. This dialog provides two options:

   - **Database Installation**: Install the Device Data into a relational database
   - **Repository File Merge**: Update an existing file-based Device Repository

   Select **Database Installation**.

10. The "Database Connection Details" dialog is displayed.

   Example of Connection Details dialog
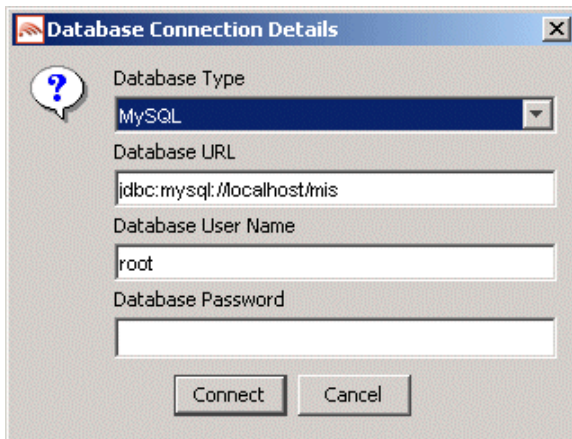
11. Select the type of database from the **Database Type** drop-down menu. Values for the **Database URL**, **Database User Name** and **Database Password** fields should be available from your Database Administrator (DBA). Enter the database connection details in the format shown in the preceding graphic and described :

   - For Oracle, enter the following in the **Database URL** field:
     jdbc:oracle:thin:@<oracle_host>:<oracle_port>:<oracle_database_name>

   - For MySQL, enter the following in the **Database URL** field:
     jdbc:mysql://<mysql-server-ip:port>/ <db-name>?user=<connect-user>&password=<connect-password>

   - For Postgres, enter the following in the **Database URL** field:
     jdbc:postgresql://<postgres_machine>:<postgres_port>/<postgres_database_name>

   - For PointBase, enter the following in the **Database URL** field:
     jdbc:pointbase:server://<ip_address>:<port>/<SID>

   - For Sybase ASE, enter the following in the **Database URL** field:
     jdbc:sybase:Tds:<ip_address>:<port>/SID

   - For IBM DB2, enter the following in the **Database URL** field:
     jdbc:db2://<ip_address>:<port>/SID

   - For Microsoft SQL Server 2000, enter the following in the **Database URL** field:
     jdbc:bea:sqlserver://<sqlserver_host>:<sqlserver_port>;databaseName=<sqlserver_database_name>

12. Click **Connect**. Once a successful connection is made the details are stored and will be remembered the next time the tool is run.

13. A progress bar shows the progress of the data installation.

    Creating Device Table Progress Bar
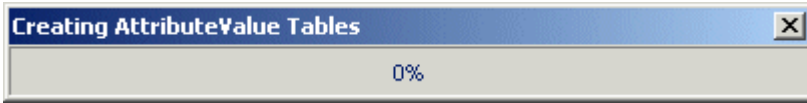
14. The "Device Repository Installation Complete" message is displayed.

    Repository Installation Complete message
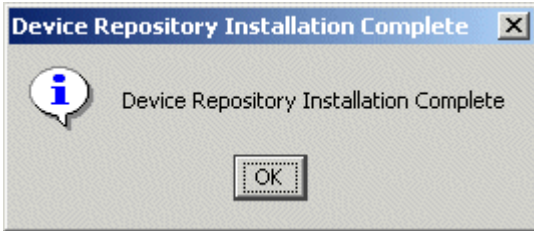
15. Click **OK**.

16. Click **Exit** on the "Device Repository Manager" screen to close the tool.

Device Repository Manager Scenario 2: Use the Online Update Service to Update the Device Repository

Follow the steps below to connect to the Online Update Service to update an existing Device Repository.

Note: You can also configure Device Repository Manager to connect to the Device Repository Online Update Service via a web proxy to download the latest updates—see Appendix D for instructions on how to do this.

1. Run *DeviceRepositoryManager.exe* (Windows) or *DeviceRepositoryManager* (UNIX/Linux platforms). The "Device Repository Options" dialog is displayed.

    Device Repository Options dialog

2. Select **Download and Install Latest Device Updates** to connect to the Online Update Service for the latest *DeviceRepository* file.

    Note: If a default license file is not found, the "License File Not Found file chooser" dialog is displayed. Here you can browse to and select a valid license file.

3. The "Database Connection Details" dialog is displayed.

   Example of Connection Details screen

4. Select the type of database from the **Database Type** drop-down menu. Values for the **Database URL**, **User Name** and **Password** fields should be available from your Database Administrator (DBA). Enter the database connection details in the format shown in the preceding graphic and described :

   - For Oracle, enter the following in the **Database URL** field:
     jdbc:oracle:thin:@<oracle_host>:<oracle_port>:<oracle_database_name>

   - For MySQL, enter the following in the **Database URL** field:
     jdbc:mysql://<mysql-server-ip:port>/ <db-name>?user=<connect-user>&password=<connect-password>

   - For Postgres, enter the following in the **Database URL** field:
     jdbc:postgresql://<postgres_machine>:<postgres_port>/<postgres_database_name>

   - For PointBase, enter the following in the **Database URL** field:
     jdbc:pointbase:server://<ip_address>:<port>/<SID>

   - For Sybase ASE, enter the following in the **Database URL** field:
     jdbc:sybase:Tds:<ip_address>:<port>/SID

   - For IBM DB2, enter the following in the **Database URL** field:
     jdbc:db2://<ip_address>:<port>/SID

   - For Microsoft SQL Server 2000, enter the following in the **Database URL** field:
     jdbc:bea:sqlserver://<sqlserver_host>:<sqlserver_port>;databaseName=<sqlserver_database_name>

5. Click **Connect**. Once a successful connection is made the details are stored and will be remembered the next time the tool is run.

6. If you successfully connect and are authorized to receive the latest *DeviceRepository* file, it will download now. The Device Browser screen then displays, which shows the downloaded *DeviceRepository* file.

Device Browser
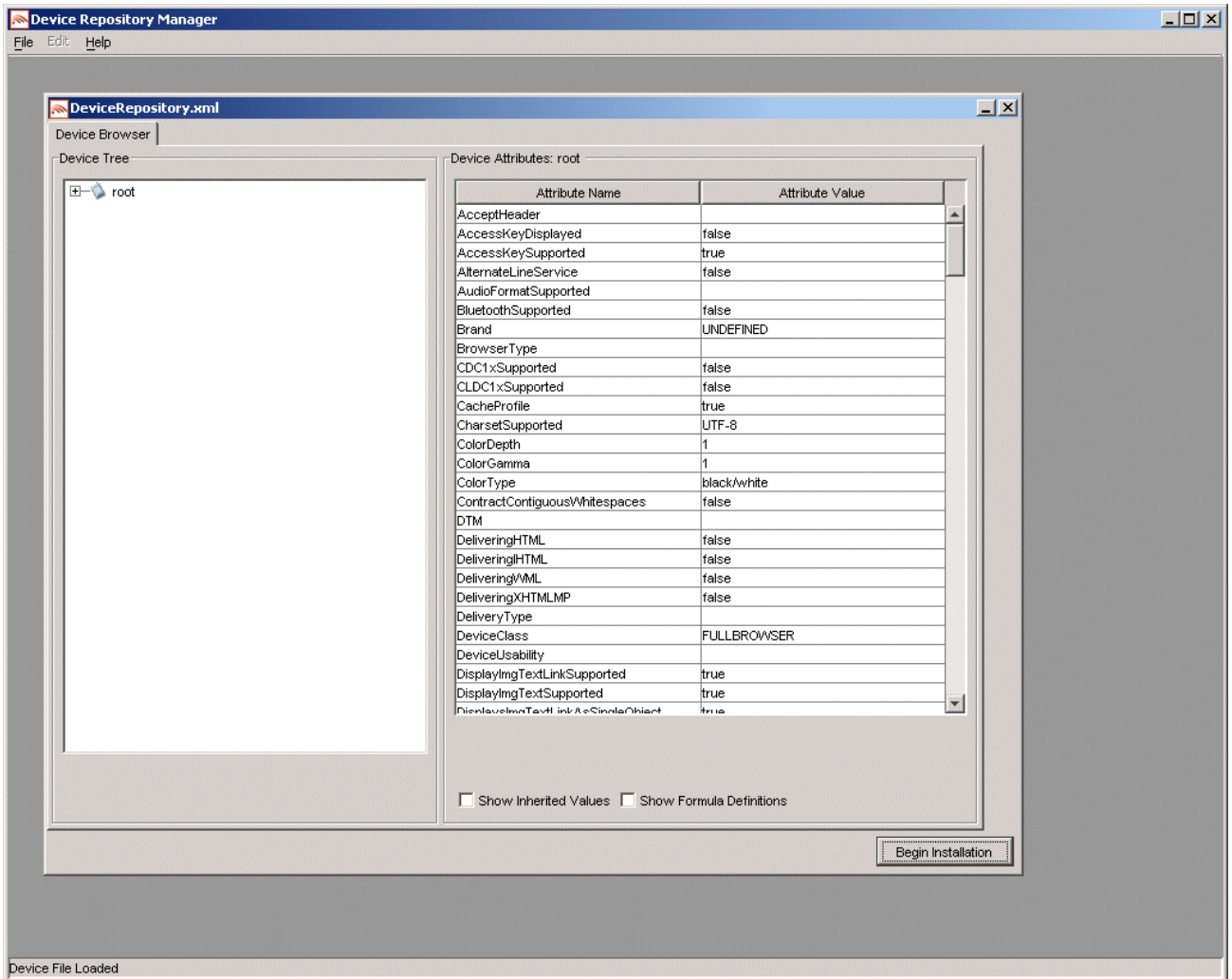
7. The data is shown in a hierarchical structure as a preview of the Device Repository to be installed. To display inherited values for each device, select the **Show Inherited Values** check box. Click **Begin Installation** to proceed.

Note: You may be required to re-enter the Database Connection details; if so, see steps 3—5.

8. If an existing Device Repository is detected, the "Backup Existing Database to file" dialog is displayed.

Backup Existing Database to File dialog
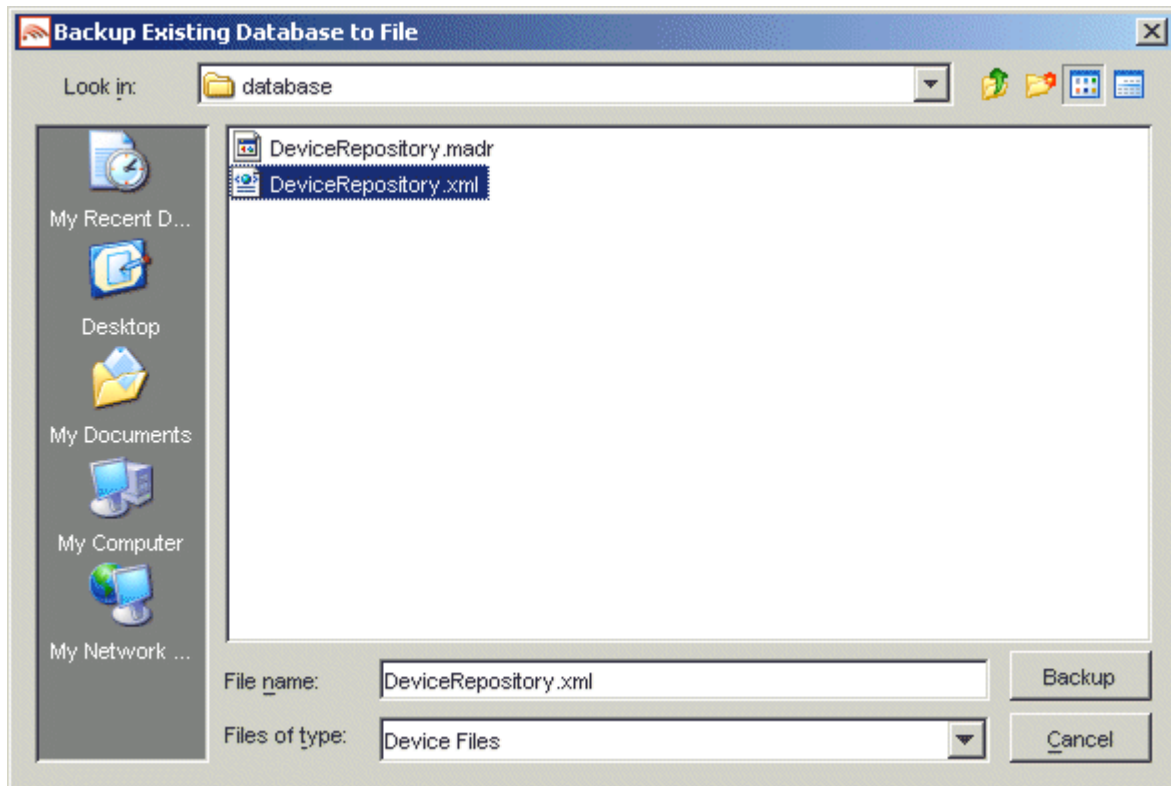
Here, you must backup the existing Device Repository to a file before proceeding. The created backup file will have the same structure as the *DeviceRepository* file.

This backup file will be used for detecting modifications later in the upgrade process.

Enter a name for the file and click **Backup**.

9. A progress bar monitors the progress of the backup process. This may take up to two minutes depending on the connection.

Backup Progress Bar
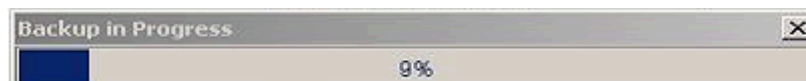
The system will now compare the *backup* and *DeviceRepository* files to compile a list of modifications.

10. If there are no modifications, the system will replace the existing database with the selected Device Repository file. In this case you can now proceed to the next step.

If it does detect modifications, you must review these modifications before proceeding with the installation. In this case, continue with this step.

Once the detection process completes, a dialog box similar to the one shown will display showing the delta between the *DeviceRepository* file to be installed and the existing Device Repository.

Comparison of Existing Download Data with New Data
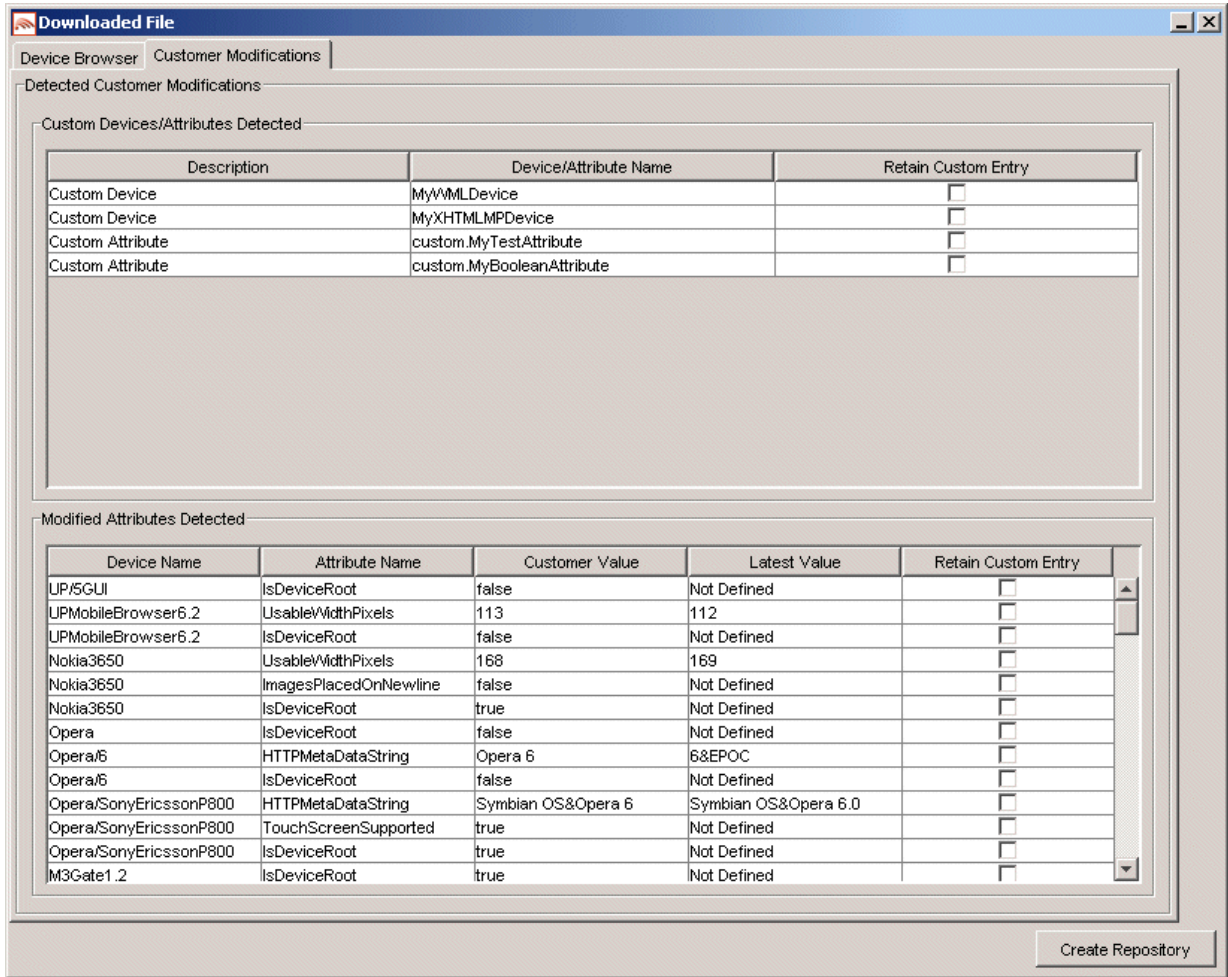
You can re-apply any modifications that have been detected in the existing Device Repository by selecting the appropriate **Retain Custom Entry** check boxes.

You can now create the database. After selecting any data that you wish to retain, click **Create Repository**.

11. A "Warning" dialog box is displayed indicating that the Device Repository data will be replaced with the selected data set.

Overwrite Data Screen
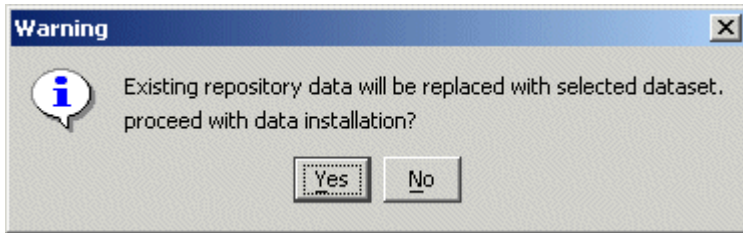
12. Click **Yes** to proceed with the installation.

13. A progress bar shows the progress of the data installation.

Creating Device Table Progress Bar
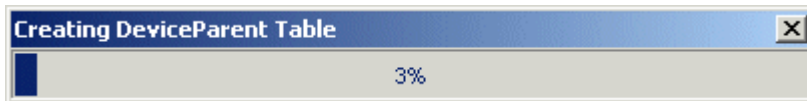
14. The "Device Repository Installation Complete" message is displayed.

Database Repository Installation Complete message

15. Click **OK** and then click **Exit** on the "Device Repository Manager" screen to close the tool.

Device Repository Manager Scenario 3: Update an Existing Device Repository from a File

Follow these steps to update an existing Device Repository from a file:

1. Run *DeviceRepositoryManager.exe* (Windows) or *DeviceRepositoryManager* (UNIX/Linux platforms). The "Device Repository Manager Usage" dialog is displayed. Click **Continue**.

2. The "Device Repository Options" dialog is displayed.

    Device Repository Options

3. Select **Install/Update Device Repository from File** to update the Device Repository using a local *DeviceRepository*.

4. The "Select Latest Device Repository file" dialog is displayed.

    Select Latest Device Repository File dialog

5. Select a *DeviceRepository* file to install and click **Open**. The *DeviceRepository* file included with the WebLogic Mobility Server installer will be shown as the default for a new installation.

6. The "Device Browser" screen shows the downloaded *DeviceRepository*.

   Device Browser Screen

7. The data is shown in a hierarchical structure as a preview of the Device Repository to be installed. To display inherited values for each device, select the **Show Inherited Values** check box. Click **Begin Installation** to proceed.

8. The second "Device Repository Options" dialog is displayed.

    Second Device Repository Options dialog

    It provides two options:

    - **Database Installation**: Install the Device Data into a relational database
    - **Repository File Merge**: Update an existing file-based Device Repository

    Select **Database Installation**.

9. The "Database Connection Details" dialog is displayed.

    Example of Connection Details screen

10. Select the type of database from the **Database Type** drop-down menu. Values for the **Database URL**, **User Name** and **Password** fields should be available from your Database Administrator (DBA). Enter the database connection details in the format shown in the preceding graphic and described :

    - For Oracle, enter the following in the **Database URL** field:
      jdbc:oracle:thin:@<oracle_host>:<oracle_port>:<oracle_database_name>

    - For MySQL, enter the following in the **Database URL** field:
      jdbc:mysql://<mysql-server-ip:port>/ <db-name>?user=<connect-user>&password=<connect-password>

    - For Postgres, enter the following in the **Database URL** field:
      jdbc:postgresql://<postgres_machine>:<postgres_port>/<postgres_database_name>

    - For PointBase, enter the following in the **Database URL** field:
      jdbc:pointbase:server://<ip_address>:<port>/<SID>

    - For Sybase ASE, enter the following in the **Database URL** field:
      jdbc:sybase:Tds:<ip_address>:<port>/SID

    - For IBM DB2, enter the following in the **Database URL** field:
      jdbc:db2://<ip_address>:<port>/SID

    - For Microsoft SQL Server 2000, enter the following in the **Database URL** field:
      jdbc:bea:sqlserver://<sqlserver_host>:<sqlserver_port>;databaseName=<sqlserver_database_name>

11. Click **Connect**. Once a successful connection is made the details are stored and will be remembered the next time the tool is run.

12. If an existing Device Repository is detected, a "Backup Existing Database to file" dialog is displayed.

    Backup Existing Database to File dialog
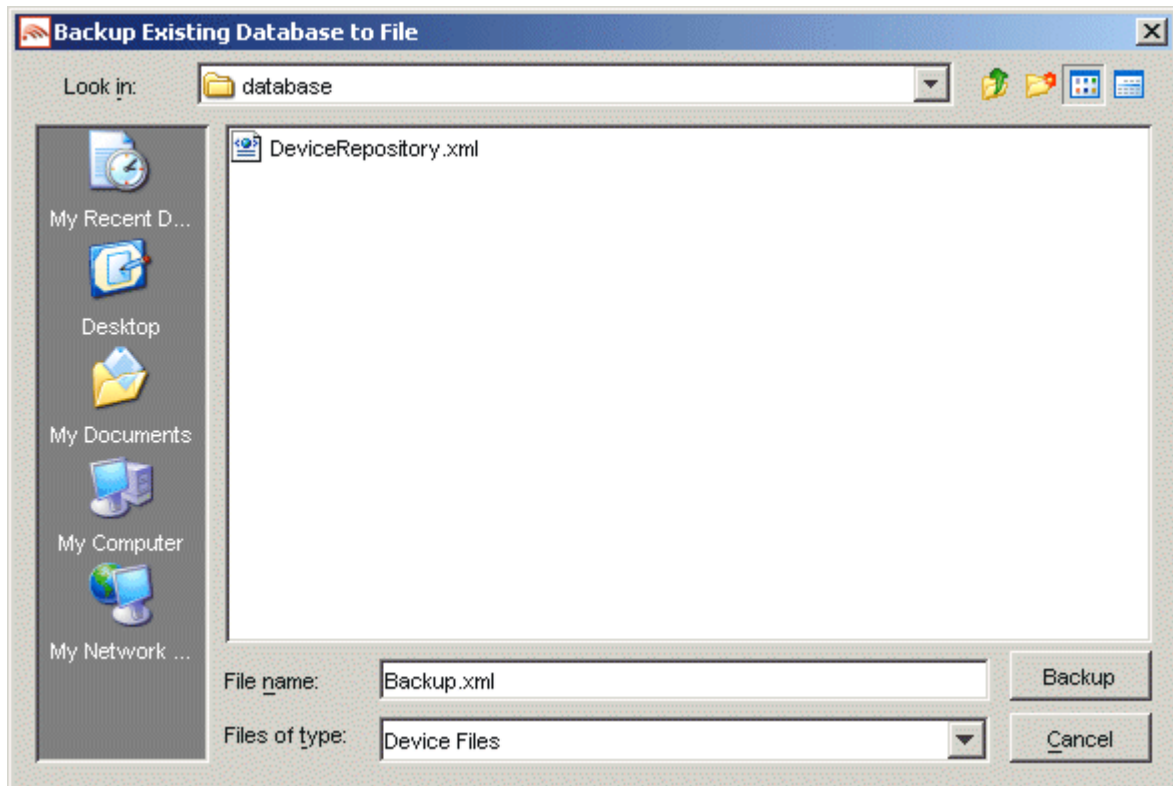
13. Here, you must backup the existing Device Repository to a file before proceeding. The created backup file will have the same structure as the *DeviceRepository* file. This backup file will be used for detecting modifications later in the upgrade process. Enter a name for the file and click **Backup**.

14. A progress bar monitors the progress of the backup process. This may take up to two minutes depending on the connection.

    Backup Progress Bar
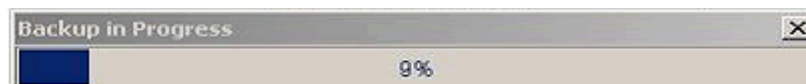
    The system will now compare the *backup* and *DeviceRepository* files to compile a list of modifications.

15. If there are no modifications, the system will replace the existing database with the selected *DeviceRepository* file. In this case you can proceed now to the next step.

    If it does detect modifications, you must review these modifications before proceeding with the installation. In this case, continue with this step.

    Once the detection process completes, a dialog box similar to the one shown will display showing the delta between the *DeviceRepository* file to be installed and the existing Device Repository.

Comparison of Existing Data with New Data

You can now choose to re-apply any modifications that have been detected in the existing Device Repository by selecting the appropriate **Retain Custom Entry** check boxes.

You can now create the database. After selecting any data that you wish to retain, click **Create Repository**.

16. A "Warning" dialog box is displayed indicating that the Device Repository data will be replaced with the selected data set.

Overwrite Data Screen
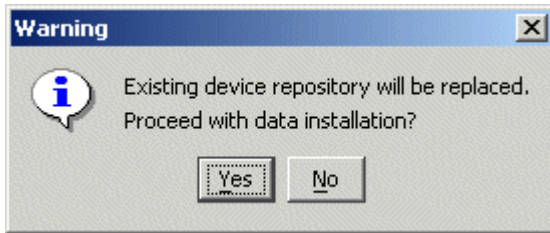
17. Click **Yes** to proceed with the installation.

18. A progress bar shows the progress of the data installation.

Creating Device Table Progress Bar
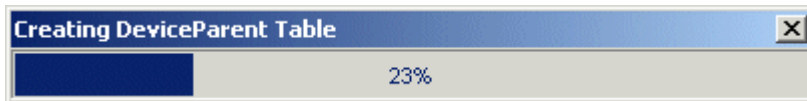
19. The "Database Repository Installation Complete" message is displayed.

Database Repository Installation Complete message

20. Click **OK** here and then click **Exit** on the "Device Repository Manager" screen to close the tool.

Device Repository Manager Scenario 4: Backup an Existing Database to a file

Follow the steps outlined to backup an existing database.

1. Run *DeviceRepositoryManager.exe* (Windows) or *DeviceRepositoryManager* (UNIX/Linux platforms).The "Device Repository Manager Usage" dialog is displayed. Click **Continue**.

2. The "Device Repository Options" dialog is displayed.

    Device Repository Options dialog

3. Select **Backup Existing Device Repository** to backup the installed database to a file.

4. The "Database Connection Details" dialog is displayed.

    Example of Connection Details screen

5. Select the type of database from the **Database Type** drop-down menu. Values for the **Database URL**, **User Name** and **Password** fields should be available from your Database Administrator (DBA). Enter the database connection details in the format shown in the preceding graphic and described :

    - For Oracle, enter the following in the **Database URL** field:
      jdbc:oracle:thin:@<oracle_host>:<oracle_port>:<oracle_database_name>

    - For MySQL, enter the following in the **Database URL** field:
      jdbc:mysql://<mysql-server-ip:port>/ <db-name>?user=<connect-user>&password=<connect-password>

    - For Postgres, enter the following in the **Database URL** field:
      jdbc:postgresql://<postgres_machine>:<postgres_port>/<postgres_database_name>

    - For PointBase, enter the following in the **Database URL** field:
      jdbc:pointbase:server://<ip_address>:<port>/<SID>

    - For Sybase ASE, enter the following in the **Database URL** field:
      jdbc:sybase:Tds:<ip_address>:<port>/SID

    - For IBM DB2, enter the following in the **Database URL** field:
      jdbc:db2://<ip_address>:<port>/SID

    - For Microsoft SQL Server 2000, enter the following in the **Database URL** field:
      jdbc:bea:sqlserver://<sqlserver_host>:<sqlserver_port>;databaseName=<sqlserver_database_name>

6. Click **Connect**. Once a successful connection is made the details are stored and will be remembered the next time the tool is run.

7. The "Backup Existing Database to file" dialog is displayed.

    Backup Existing Database to File dialog
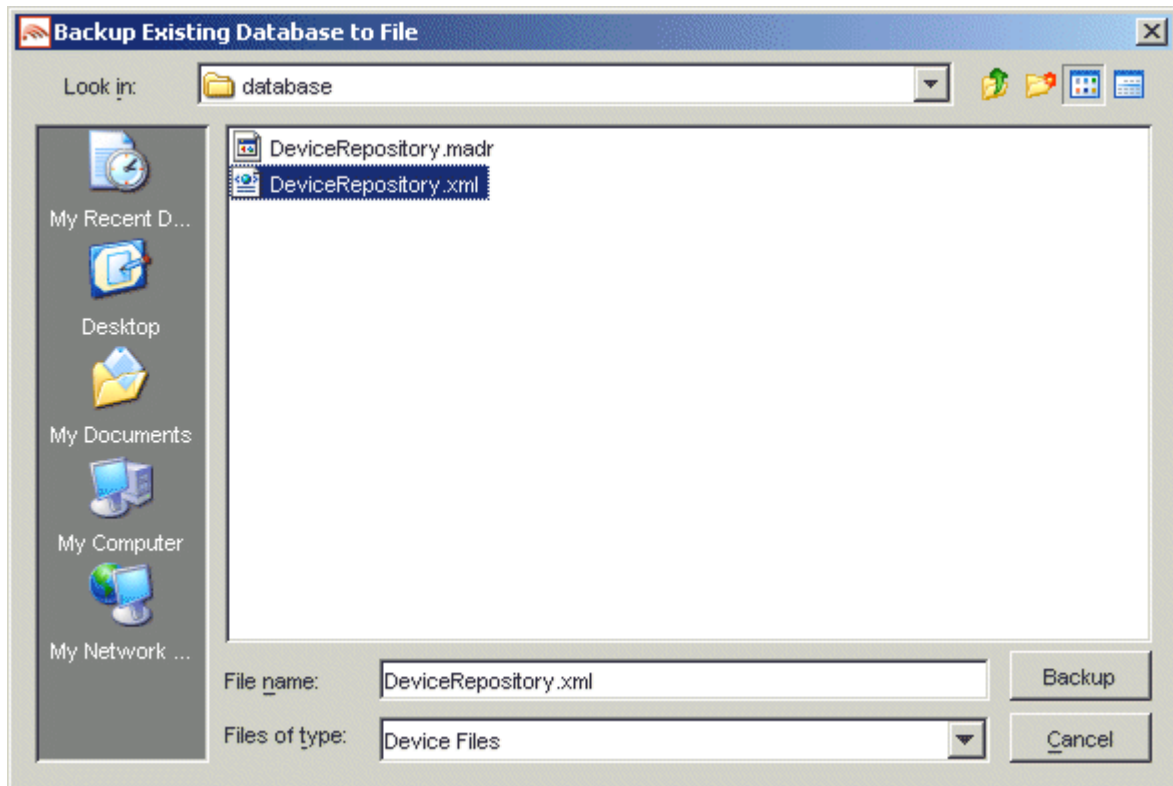
8. Enter a name for the file and click **Backup** to backup the existing Device Repository to a file.

9. A progress bar monitors the progress of the backup process. This may take up to two minutes depending on the connection.

    Progress Bar
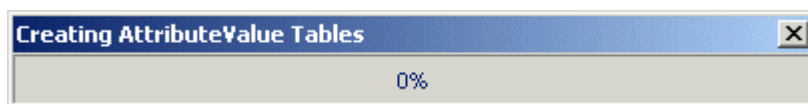
10. When the backup completes, click **Exit** on the "Device Repository Manager" screen to close the tool.

Device Repository Manager Scenario 5: Update a File-Based Device Repository

If a customized file-based Device Repository already exists, you can use Device Repository Manager to merge these customizations into the latest *DeviceRepository* file.

Note: It is recommended that you backup the customized *DeviceRepository* file before proceeding with the update process.

Follow the steps outlined to update an existing file-based Device Repository:

1. Run *DeviceRepositoryManager.exe* (Windows) or *DeviceRepositoryManager* (UNIX/Linux platforms). The "Device Repository Manager Usage" dialog is displayed. Click **Continue**.

2. The "Device Repository Options" dialog is displayed.

   Device Repository Options dialog

3. Select **Install/Update Device Repository from File** to update the Device Repository using a local *DeviceRepository*.

4. The "Select Device Repository file" dialog is displayed.

    Select Latest Device Repository File dialog

5. Select a *DeviceRepository* file to install and click **Open**. The *DeviceRepository* file included with the WebLogic Mobility Server installer will be shown as the default for a new installation.

6. The following "Device Browser" screen shows the downloaded *DeviceRepository*.
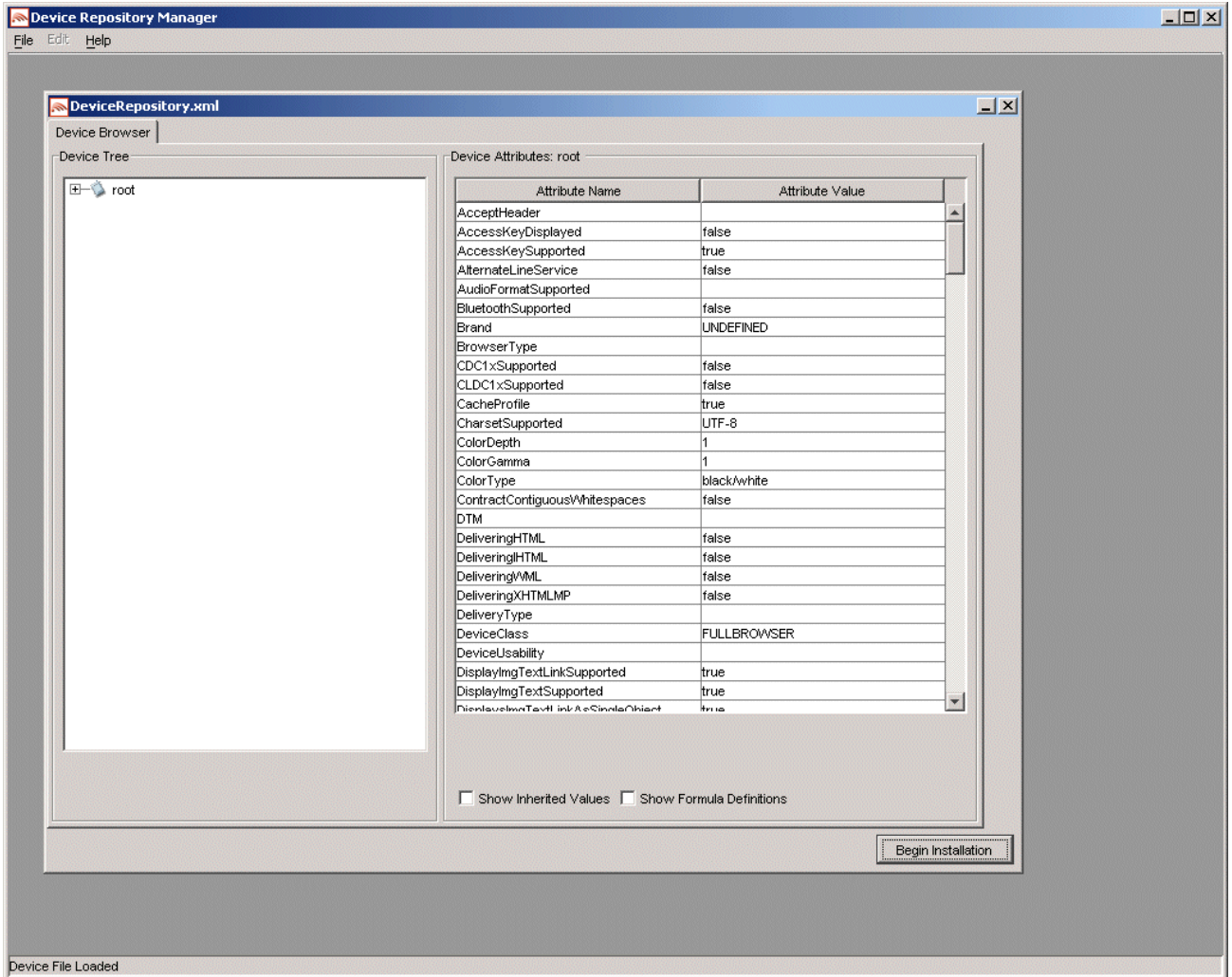
    Device Browser Screen

7. The data is shown in a hierarchical structure as a preview of the Device Repository to be installed. To display inherited values for each device, select the **Show Inherited Values** check box. Click **Begin Installation** to proceed.

8. The second "Device Repository Options" dialog is displayed.

    It provides two options:

    - **Database Installation**: Install the Device Data into a relational database
    - **Repository File Merge**: Update an existing file-based Device Repository

    Second Device Database Options screen
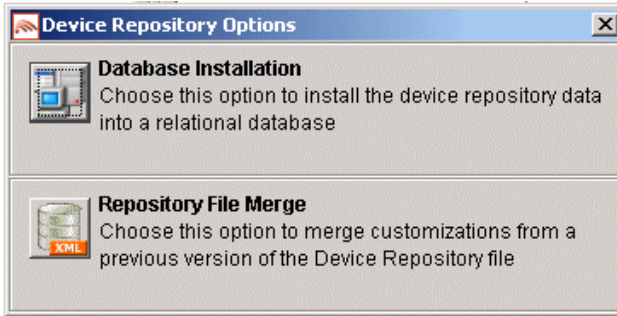
    Select **Repository File Merge**.

9. The "Select Customized Device File" dialog is displayed.

    Select Customized Device File dialog
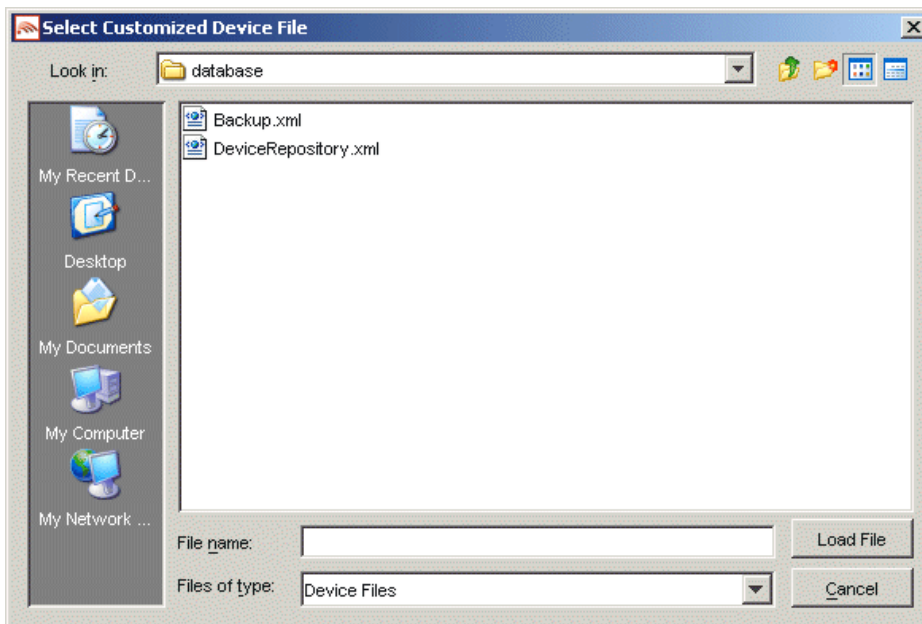
    Verify that the correct customized file is selected and click **Load file**.

10. If:

- No modifications are detected the "File Merge Not Required" message is displayed

File Merge Not Required screen

File Merge Not Required

No customer modifications were detected
File Merge is not required, the Device Repository Manager will now exit.
Check that the correct files have been loaded.

[OK]
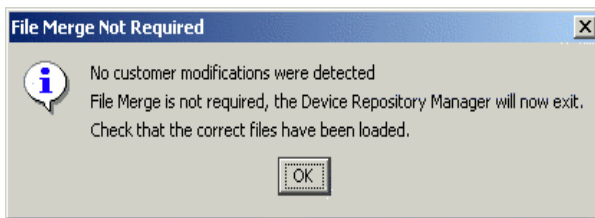
Click **OK** to exit the tool.

- If modifications are detected, the "Customer Modifications" screen is displayed

Customer Modifications screen

Downloaded File

Device Browser | Customer Modifications
Detected Customer Modifications
Custom Devices/Attributes Detected

Description	Device/Attribute Name	Retain Custom Entry
Custom Device	MyWMLDevice	☐
Custom Device	MyXHTMLMPDevice	☐
Custom Attribute	custom.MyTestAttribute	☐
Custom Attribute	custom.MyBooleanAttribute	☐

Modified Attributes Detected

Device Name	Attribute Name	Customer Value	Latest Value	Retain Custom Entry
UP/5GUI	IsDeviceRoot	false	Not Defined	☐
UPMobileBrowser6.2	UsableWidthPixels	113	112	☐
UPMobileBrowser6.2	IsDeviceRoot	false	Not Defined	☐
Nokia3650	UsableWidthPixels	168	169	☐
Nokia3650	ImagesPlacedOnNewline	false	Not Defined	☐
Nokia3650	IsDeviceRoot	true	Not Defined	☐
Opera	IsDeviceRoot	false	Not Defined	☐
Opera/6	HTTPMetaDataString	Opera 6	6&EPOC	☐
Opera/6	IsDeviceRoot	false	Not Defined	☐
Opera/SonyEricssonP800	HTTPMetaDataString	Symbian OS&Opera 6	Symbian OS&Opera 6.0	☐
Opera/SonyEricssonP800	TouchScreenSupported	true	Not Defined	☐
Opera/SonyEricssonP800	IsDeviceRoot	true	Not Defined	☐
M3Gate1.2	IsDeviceRoot	true	Not Defined	☐

Create Repository

Select the corresponding **Retain Custom Entry** check boxes for any attributes/devices that you want to keep and click **Create Repository**.

11. The "Export File" dialog is displayed.

Export File dialog
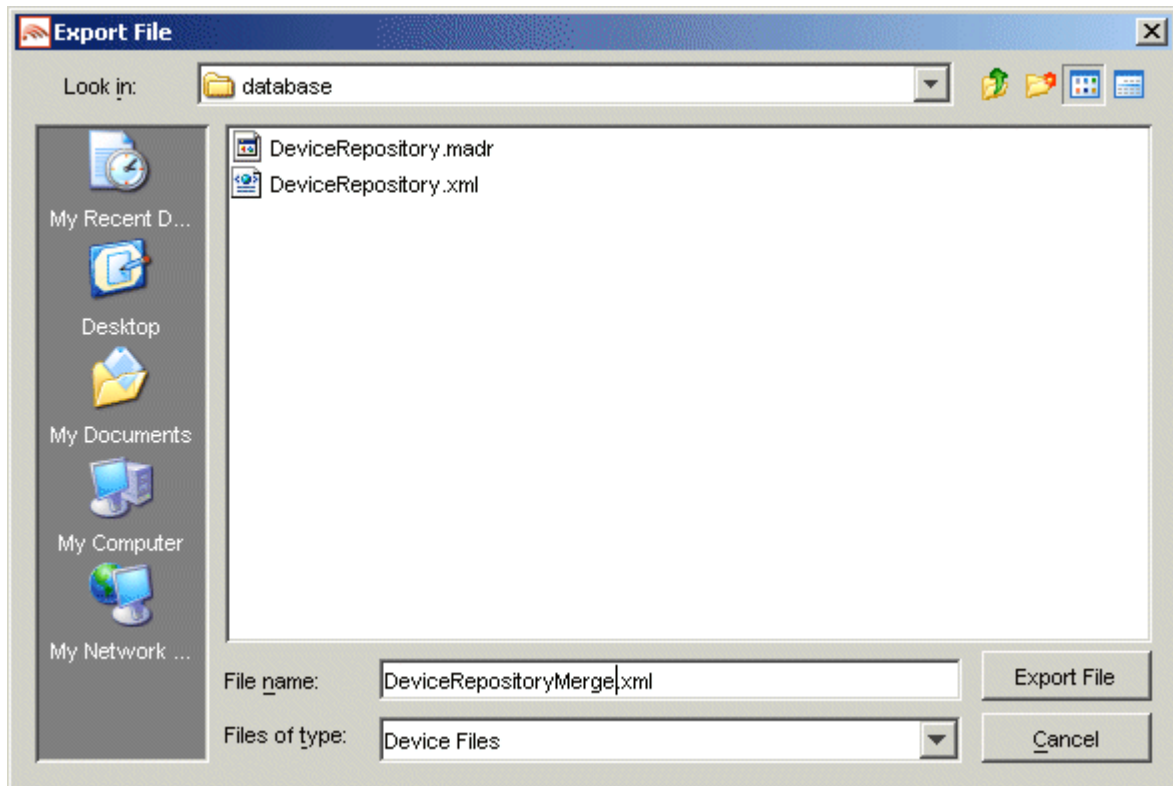
12. Select the filename for the newly merged file.

    Notes

    - You may use an existing file; however, it is advisable to use a new filename

    - Files with ".madr" extensions contain compressed device repositories. If you specify a ".madr" file extension, the file that you create will be a compressed version of the Device Repository

13. When the file has been written to disk, click **Exit** to exit the tool.

2—Configure the mis.properties Settings

The *mis.properties* file is a configuration file containing the Device Repository connection settings used by WebLogic Mobility Server. You must modify the file to reflect the Device Repository connection details, so that WebLogic Mobility Server can connect to the Device Repository and retrieve device profiles.

Locate the mis.properties File

The *mis.properties* file is a plain text file that can be edited in any text editor. The file can be found in the **WEB-INF/classes** folder of the web application.

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 *DeviceRepository* file, follow the instructions in the "Configure a File-Based Device Repository" section.

Configure a Database Device Repository

You must configure the following Device Repository properties in order for WebLogic Mobility Server to successfully communicate with the Device Repository when using an external database:

Device Repository properties settings

Property	Description
deviceDB.driver	This is the location of the JDBC driver that WebLogic Mobility Server will use to gain access to the database.
	This property also has the effect of informing WebLogic Mobility Server of the database that 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	This is the URL used 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>
	Notes
	• For MySQL 3.X, set to: jdbc:mysql://<mysql-server-ip:port>/ <db-name>?user=<connect-user>&password=<connect-password>
	• For MySQL 4 or 5, set to: jdbc:mysql://<mysql-server-ip:port>/ <db-name>
	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, 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	This is the username that WebLogic Mobility Server uses to access the database server when user and password authentication is required.
	Note: For MySQL 3.X, this property is left clear.
	Example: deviceDB.user: user
deviceDB.password	This is the password that WebLogic Mobility Server uses to access the database server when user and password authentication is required.
	Note: For MySQL 3.X, this property is left clear.
	Example: deviceDB.password: password
deviceDB.maxDBConnections	This is 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. The default is "10". For more information, see the section "About Connection Pools".
	Example: deviceDB.maxDBConnections: 10
deviceDB.waitTime	This is a numeric value indicating (in milliseconds) the waiting time for a

	connection from the database pool. Defaults to 5000.
	Example: deviceDB.waitTime: 5000
deviceDB.increment	This is 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. The default is "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 and 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.

You may deploy the *DeviceRepository* file in one of two ways:

- In an absolute location

- On the CLASSPATH

Deploy the DeviceRepository file in an Absolute Location

1. Locate the *mis.properties* file for your web application (for example, for WebLogic Mobility Server look in the **WEB-INF/classes** folder of the web application). Open it in a text editor.

2. 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
```

Change the last line so that it now reads:

```
deviceRepositoryType: xml
```

3. 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
#
# Example:
#C:/WebLogic Mobility Server/database/DeviceRepository.madr
#
#deviceXML.location: <install_directory>/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** (WebLogic Mobility Server).

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

4. Save the *mis.properties* file.

5. In a production environment, you must now re-deploy or re-start your web applications.

Deploy the DeviceRepository File on the CLASSPATH

1. Either add the directory containing the *DeviceRepository* file to the CLASSPATH, or deploy the *DeviceRepository* 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, for a sample News application, it 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; see example:

```
######################################################
#
# 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.

Next steps

Proceed to the *BEA WebLogic Mobility Server Administration Guide* and follow the instructions there to configure and manage WebLogic Mobility Server.

When administering the Device Repository at a later stage, you may find it useful to see the next chapter of this guide, which describes how to set up and manage the device profiles stored in the database.

3—Administer the Device Repository

Introduction

Mobile devices have a range of different input and presentation capabilities, network connectivity and levels of scripting language support.

WebLogic Mobility Server accommodates these differences by maintaining a Device Repository, which contains profiles describing the properties and capabilities for a range of devices on the market.

These device profiles enable WebLogic Mobility Server to tailor the presentation and delivery of content to each device. This ensures that clients receive content that they can display and store, and which doesn't take too long to convey over the network.

Note: For more information on how WebLogic Mobility Server uses the Device Repository, see "Appendix A"

This chapter explains how to set up and manage the device profiles stored in the Device Repository. To do this you will use the Device Repository Manager tool, which enables you to conveniently set up, retrieve and modify the various attributes associated with each profile.

Important note: When the Device Repository is represented as a database, you will use the Admin Console tool to add, remove and modify devices and device attributes; for more information, see "Appendix C".

More About Device Profiles

Each device profiled in the Device Repository has an associated set of properties (attribute-value pairs) that enable WebLogic Mobility Server to identify the requesting device in order to deliver and present the content appropriately. In the event that WebLogic Mobility Server does not find an exact match within its profiles, it uses the attributes to determine the closest match.

Composite Capabilities/Preferences Profile (CC/PP) is a standard developed by the W3C that is used to describe device capabilities and user preferences (i.e. the delivery context). This information can be used to develop device independent web content or applications. 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 standard has been adopted for the Device Repository. Currently, the Repository is CC/PP compliant, containing both the UAProf attribute set and a more comprehensive set of WebLogic Mobility Server proprietary device properties.

Each device is described by a set of attributes that make up a unique profile for that device. Both types of attributes are described here.

CC/PP Attributes

Following the standard, the CC/PP compliant attributes fall into one of seven categories. Each attribute begins with a prefix that indicates into which category it falls. The following table lists these categories and gives examples of the types of attributes that they encompass.

CC/PP attribute category prefixes and example attributes

Category prefix	Example attributes
UAProf.BrowserUA	BrowserName FramesCapable HtmlVersion TablesCapable
UAProf.HardwarePlatform	ScreenSize ColorCapable ImageCapable Vendor
UAProf.MmsCharacteristics	MmsCcppAccept MmsMaxImage
UAProf.NetworkCharacteristics	SupportedBluetoothVersion SecuritySupport
UAProf.PushCharacteristics	Push-Accept-Charset Push-Accept-Language
UAProf.SoftwarePlatform	OSName OSVendor VideoinputEncode
UAProf.WapCharacteristics	WmlScriptLibraries WapVersion WmlDeckSize

Proprietary Attributes

The proprietary attributes describe device characteristics that are not yet included in the standard, but describe a number of extra characteristics that can be used when tailoring content to particular devices.

Sample Proprietary Attributes

Attribute name
AccessKeySupported
FlashSupported
RingtoneDownloadSupported
IsMenuDriven

More About the Device Repository

The Organization of the Device Profiles

The Device Repository represents devices as a hierarchical arrangement, thus enabling devices to inherit attributes from a parent device.

The Device Repository has three parent device classes: WML, XHTMLMP and HTML devices. When adding a new device, you can place it within one of these hierarchies or create your own parent device class.

The tree is branched on the markup language used by the device, with HTML, WML and XHTMLMP providing the main branches off the default "root". The main branches, devices are categorized according to browser or model type, as illustrated in the device hierarchy.

Example Device Repository hierarchy fragment
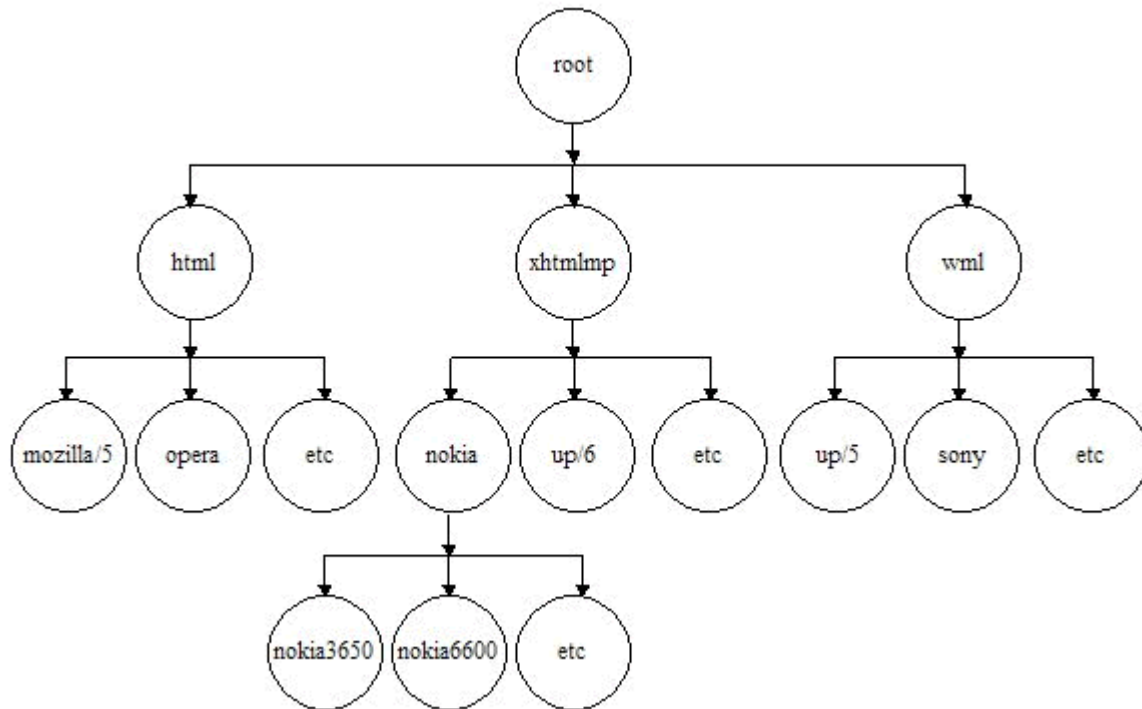
The hierarchy stores device attributes for the named devices detailing markup languages, screen sizes, and so on.

Device attribute inheritance
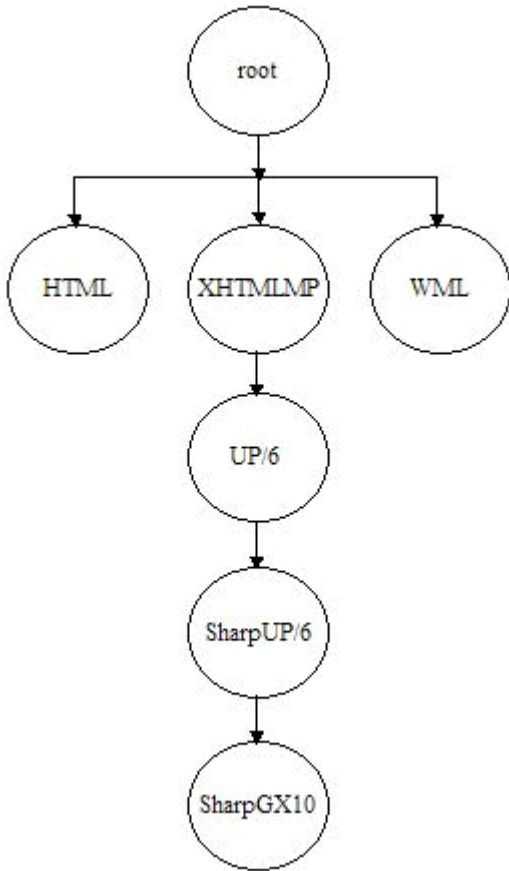
A node in the device tree inherits any device attributes (markup language, screen size, and so on.) from its parent. If the child node specifies values for any of these attributes, the child's values override those of its parents.

Device Recognition

Compare Incoming Requests to Device Attributes

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 markup language of the device and often provides device model information) and stored device attributes.

WebLogic Mobility Server achieves this by examining the details of the request and matching this request against device attributes contained in the Device Repository.

The pattern matcher will firstly attempt to match on the RecognitionUAPattern stored device attribute; if there is no match here it will subsequently attempt to match on the HTTPMetadataKey attribute—both mechanisms are described in more detail below.

1. RecognitionUAPattern Match Mechanism

The pattern matcher will firstly attempt to match values in the User-Agent header information from the incoming request with the RecognitionUAPattern attribute in the Device Repository. This attribute defines the regular expression or string to be matched against the User-Agent header.

If there is a match here, the pattern matcher will move on to determine whether the RecognitionHeaders attribute has been set or not for specific headers in the incoming request—see below.

No Match

If a match is not found, the pattern matcher will subsequently attempt to match on the HTTPMetadataKey attribute—see the "HTTPMetadataKey Match Mechanism" section.

Note: Regular Expressions

If you intend for WebLogic Mobility Server to interpret the RecognitionUAPattern attribute and the pattern component of the RecognitionHeaders attribute as regular expressions, then you must set the "RecognitionRequiresRegex" attribute to "true".

Examples of Regular Expressions

Regular Expression	Description
.*exampleText.*	This regex will match anything containing the string "*exampleText*".
.*exampleText$	This regex will match anything ending with the string "*exampleText*".
^exampleText.*	This regex will match anything beginning with the string "*exampleText*".

RecognitionHeaders Attribute Match

Not all information required for device recognition may be contained with the User-Agent header therefore the RecognitionHeaders attribute can contain a list of additional headers that can be checked to achieve as accurate a match as possible.

If the RecognitionHeaders attribute has *not* been set for specific headers in the incoming request, then an initial match is achieved at this point.

If it *has* been set, the pattern matcher will attempt to search and match on the additional headers listed in the RecognitionHeaders attribute. Once all headers are matched, an initial match is achieved here.

Once an initial match has been achieved at this point, the pattern matcher will move on to check whether the initially-matched device is associated with other devices that have the RecognitionCheckMeBefore attribute set.

RecognitionCheckMeBefore Attribute Match

A device may match more than one pattern. For example, the User-Agent "Ericsson t68i" would match the pattern "t68i", but it would also erroneously match the pattern "t68". The User-Agent "Ericsson t68" would only match the pattern "t68". Therefore, the pattern "t68i" must be checked first and if this fails to match, *then* the other pattern may be checked. The RecognitionCheckMeBefore attribute specifies a list of devices that a given device should take precedence over in the recognition process.

If the initially-matched device is associated with other devices that have the RecognitionCheckMeBefore attribute set, the pattern matcher will check these devices for a closer match. If there is no closer match then a full match is reached at this stage. If there *is* a closer match, then this new match will take precedence and be selected as the full match. This thorough verification ensures that an accurate in-depth match is achieved.

2. HTTPMetadataKey Match Mechanism

If a match is not achieved against the RecognitionUAPattern attribute, the pattern matcher will subsequently attempt to match on the HTTPMetadataKey attribute.

In matching a user request against a device, each level of the tree is traversed. The pattern matcher starts at the root node and attempts to traverse the tree to as deep a level (hence as specific a device match) as possible.

Each node in the tree specifies a single header and associated values that are used to differentiate it from its parent node.

The HTTPMetadataKey attribute tells the matcher to match on either Accept header or User-Agent string.

The pattern matcher will determine from the Accept header whether it is to traverse the WML branch, the XHTMLMP branch, or the HTML branch of the device hierarchy. This is defined in the HTTPMetaDataString.

Note: The ordering of child nodes is important, as the pattern matcher will take the first match found and ignore all others.

Multiple Header Strings

If more than one string must be present in the header, pattern matching is achieved either by using more than one level of the hierarchy with one of the strings specified in each or by combining the strings in a single node with an ampersand ("&") character. For example, a menu-driven device could require that two strings be matched: "text/vnd.wap.wml&image/vnd.wap.wbmp". Similarly, an "OR comparison" may be performed using the bar ("|") character.

No Match

If an absolute match is not found, a more general match is found at a higher level so that WebLogic Mobility Server can deliver content in some format understandable by the given device.

If an unknown device sends a request, WebLogic Mobility Server will find the closest match possible in the existing hierarchy (for example, an unknown UP 6.x browser-based phone will still match as far as UP 6.x). As such, WebLogic Mobility Server does not need to have an exhaustive list of all devices on the market at present.

Fallback Recognition Logic Feature

To enable the pattern matcher to consider more than one header during device recognition at any given node, you can set the FallbackRecognitionLogic attribute.

Any existing values here will override the HTTPMetaDataKey and HTTPMetaDataString attributes to allow a more advanced mechanism for determining whether or not a node should be matched during device recognition. The attribute allows multiple headers to be considered during the recognition process.

The fallback recognition logic feature is particularly useful at the top-level WML and XHTMLMP nodes as the Accept header alone may not give enough information to decide which of these nodes (if either) is the correct one to choose. If the device making the request is not known in the database (which is most likely the case at this point as otherwise the device would probably have been matched by the new UAPattern algorithm), choosing which of these nodes/sub-trees to use is the most important decision in the recognition process as it will determine whether WML or XHTMLMP markup is sent to the device.

Please see "Appendix E—Fallback Recognition Logic Expression Language Details" for more information on the FallbackRecognitionLogic attribute's associated expression language.

Important note: Like any other attribute, the FallbackRecognitionLogic expression will be inherited by child nodes, which is unlikely to be the intended behaviour. Therefore if child nodes do not have their own recognition logic expression, they should be given the special value of "none" for this attribute. In particular, all direct children of the WML and XHTMLMP nodes should initially be given a FallbackRecognitionLogic value of "none".

Device Recognition Examples

RecognitionUAPattern Match Example

The Device Repository is organized in a hierarchical (tree) structure. WebLogic Mobility Server traverses the tree to find the device that matches the received headers. WebLogic Mobility Server will linearly compare each device until a match is found.

In the example below we'll look at a simple scenario whereby the Openwave SDK 6.2.2 Emulator device sends an incoming request.

Example—Device: Openwave SDK 6.2.2 Emulator

User-Agent:

OPWV-SDK/62 UP.Browser/6.2.2.1.208 (GUI) MMP/2.0

Example of Unsuccessful Match

Note: For simplicity's sake, assume that the RecognitionRequiresRegex attribute is "false" for all devices.

Let's take a sample scenario whereby the pattern matcher attempts to match the incoming request against a device named "Nokia7250", which has a RecognitionUAPattern of "Nokia7250".

The requesting device (the Openwave emulator) sends a User Agent of: OPWV-SDK/62 UP.Browser/6.2.2.1.208 (GUI) MMP/2.0

WebLogic Mobility Server will search the User Agent header from the incoming request for an instance of the RecognitionUAPattern of the device it is currently checking against (i.e. the "Nokia7250").

Since "Nokia7250" does not appear in "OPWV-SDK/62 UP.Browser/6.2.2.1.208 (GUI) MMP/2.0", this device does not match and WebLogic Mobility Server will move on to the next device.

Example of Successful Match

Let's take a sample scenario whereby the pattern matcher attempts to match the incoming request against a device named "UPMobileBrowser6.2", which has a RecognitionUAPattern of "OPWV-SDK/62".

The requesting device (the Openwave emulator) sends a User Agent of: OPWV-SDK/62 UP.Browser/6.2.2.1.208 (GUI) MMP/2.0

WebLogic Mobility Server searches the User Agent header from the incoming request for an instance of the RecognitionUApattern of the device it is currently checking against (i.e. the "UPMobileBrowser6.2").

Accept:

User-Agent:OPWV-SDK/62 UP.Browser/6.2.2.1.208 (GUI) MMP/2.0

OPWV-SDK/62

As can be seen above, "OPWV-SDK/62" does appear in "OPWV-SDK/62 UP.Browser/6.2.2.1.208 (GUI) MMP/2.0" therefore an initial match is achieved with this device.

As explained in section "RecognitionHeaders Attribute Match" WebLogic Mobility Server will now move on to examine the Recognition Headers—if present.

If RecognitionHeaders match, WebLogic Mobility Server will proceed to check devices that specify "UPMobileBrowser6.2" in their respective RecognitionCheckMeBefore fields—see section "RecognitionCheckMeBefore Attribute Match" for a further explanation of this process.

HTTPMetaDataKey Match Example

The Device Repository is organized in a hierarchical (tree) structure. WebLogic Mobility Server traverses the tree to find the device that matches the received headers. At each level in the hierarchy a different sub-string of the HTTPMetaDataString must be matched. When no more sub-strings can be matched the selected device is returned.

Example

User-Agent:

SHARP-TQ-GX10/0.0 Profile/MIDP-1.0 Configuration/CLDC-1.0 UP.Browser/6.1.0.3.107 (GUI) MMP/1.0

Accept:
application/vnd.wap.wmlc,application/vnd.wap.wmlscriptc,application/vnd.wap.multipart.related,application/
vnd.wap.multipart.mixed,application/vnd.phonecom.mmc-wbxml,application/octet-
stream,application/vnd.openwave.pp,text/plain,text/css,image/bmp,image/gif,image/jpeg,image/png,image/vn
d.wap.wbmp,image/x-up-
png,application/vnd.wap.sic,application/vnd.wap.slc,application/vnd.wap.coc,application/vnd.wap.xhtml+xm
l,**application/xhtml+xml**;profile="http://www.wapforum.org/xhtml",text/html,text/vnd.sun.j2me.app-
descriptor,application/java,application/java-archive,application/smil,application/vnd.wap.mms-
message,audio/x-wav,application/x-neva1,application/x-eva,application/x-smaf,application/vnd.smaf,text/x-
imelody,audio/x-imy,audio/imelody,audio/midi,audio/x-
midi,audio/mid,audio/wav,application/vnd.uplanet.bearer-choice-wbxml,application/x-smaf,application/x-
imy,audio/midi,text/vnd.wap.**wml**,text/vnd.wap.wmlscript,*/*;q=0.001

Level 1

WebLogic Mobility Server needs to decide on which branch of the device tree to look for this phone. The
Accept header is used to determine this.

Device recognition – Level 1
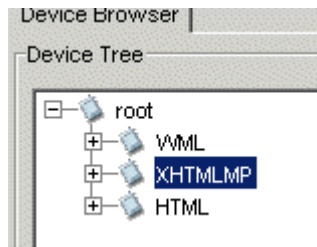
This Accept header contains **wml & xhtml+xml** so WebLogic Mobility Server will match to the XHTMLMP
branch.

WebLogic Mobility Server will now try to move further down the tree.

Level 2

WebLogic Mobility Server will now use the User-Agent string to identify the device. The User-Agent
contains **UP.Browser/6,** which WebLogic Mobility Server will match to the UP/6 branch.

Device recognition – Level 2
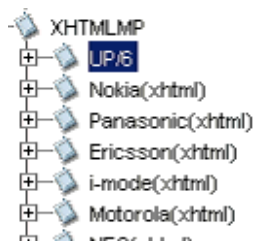
Level 3

WebLogic Mobility Server now moves into the UP/6 branch to attempt to further identify the device. The
User-Agent contains **SHARP**, which WebLogic Mobility Server will match to SharpUP/6.

Device recognition – Level 3
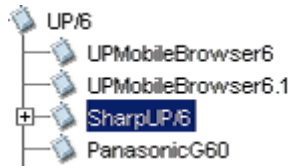
Level 4

WebLogic Mobility Server now moves into the SharpUP/6 branch to attempt to further identify the device. The User-Agent contains **GX10**, which WebLogic Mobility Server will match to SharpGX10

Device recognition – Level 4
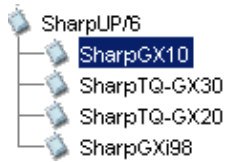
The search is now over and WebLogic Mobility Server has identified the device as Sharp GX10. Note that WebLogic Mobility Server will never look at other devices at the same level once a match has been made.

Calculated Attributes

Many of the attributes in the Device Repository are calculated from other attributes and/or incoming request headers. As these attributes use formulae they are known as formulae attributes.

Examples of the Acceptheader and ViewableWidth attributes using formulae:

Acceptheader uses toCSV(UAProf.SoftwarePlatform.CcppAccept)

ViewableWidth uses extractdimension("N",UAProf.HardwarePlatform.ScreenSize)

WebLogic Mobility Server will re-calculate the values of certain attributes for each incoming request received—these are known as dynamic attributes. Dynamic attributes use the "dynamic" formula-function (see table) and will be re-evaluated each time a request is received (—see note). This function must form the outermost element of the formula.

This ensures an accurate representation of the capabilities of devices such as the BlackBerry, which give users the ability to turn on/off table support and CSS support and are capable of modifying their browsing behaviour.

Note: If WebLogic Mobility Server needs to split a large page into smaller pages to accommodate the capabilities of the device requesting it, WebLogic Mobility Server will store the additional pages in a temporary cache specifically for pagination. WebLogic Mobility Server builds these cached pages using the status of the device requesting the first page of the paginated set. If the user changes an option on their device, e.g. tables supported, before requesting another page in the paginated set, WebLogic Mobility Server will not detect this change because the page is served from the cache. To refresh the content of the cache, the user should request a page refresh. Be aware that the requesting device may also maintain a browser cache and therefore the user may need to explicitly reload the page.

Note: *Formulae* attributes cannot be created or edited via the Device Repository Manager tool. If formulae attributes are required, please contact Customer Support.

Formulae Functions

Formula expressions can use the following functions:

Function	Description
colortype(color, bits)	Determines colortype from a true(1) or false(0) color value and number of bitsperpixel. It returns "colour", "black/white" or "greyscale".
find(substring, string)	Finds a substring in a string. It returns "true" or "false".
endswith(substring, string)	Checks if a string ends with a specified substring. It returns "true" or "false".
extractdimension(d, value)	Extracts width or height from a dimension value, i.e. "N" or "M" from "NxM".
extractformat(keyLength, key, testList)	Extracts a comma-delimited string from a list.
extractwtls(value)	Gets the WTLSSupported value from the UAProf attribute UAProf.NetworkCharacteristics.SecuritySupport
dynamic(expression)	The term "expression" denotes the dynamic formulae that MIS will calculate.
getHeader(variable)	Gets the value of a request header. For example, getHeader("Accept").
getHeaderWithDefault(variable, default)	Gets the value of a request header. Uses a default value if that header is not present in the request. **Example**: getHeaderWithDefault("Accept-charset","utf-8"). In this case MIS will use the value of the "Accept-charset" header (if present) when evaluating device capabilities. Otherwise it will use the default value (in this case "utf-8").
listContains(list,key)	Checks if a key is contained in a list. It returns "true" or "false".
select(condition, truevariable, falsevariable)	This formula calculates the boolean expression specified. If the "condition" expression evaluates to "true" MIS will calculate the "truevariable"; if it evaluates to "false", MIS will calculate the "falsevariable".
startswith(substring, string)	Checks if a string starts with a specified substring. It returns "true" or "false".
toCSV(list)	Converts a list to a comma-delimited list.

External Device Recognition API

The Device Recognition API allows you to create your own Device Recognition Classes to be invoked by the WebLogic Mobility Server external device recognition process.

Using the External Device Recognition API to Create Device Recognition Classes

1. Firstly, create the new Device Recognition Class and give it a name. Place this Class in the application's CLASSPATH. The simplest way to do this is to copy it into the application's **WEB-INF/classes** folder.

2. Then, ensure that the new Class implements the ExternalDeviceRecognizer public interface.

3. As part of this interface you will need to implement the RecognizeDevice method.
    This is the method that WebLogic Mobility Server will subsequently invoke to perform the external device recognition.

    In order for your class to compile, include the **../lib**/*mcpfilter.jar* and *servlet.jar* in your CLASSPATH. The *servlet.jar* can usually be found within your webapp server.

    Note: If WebLogic Mobility Server finds that the device has not been identified correctly, i.e. if the setDeviceID(String id) method is invoked as "setDeviceID(**null**)" or with an invalid device identifier/string during implementation of the RecognizeDevice method, then WebLogic Mobility Server will attempt its own device recognition procedure.

    Note: A warning message and diagnostic message are generated in the WebLogic Mobility Server server console and in the Diagnostics console when the setDeviceID(String id) method is invoked with an invalid device identifier.

    Example: *[MIS.Warning] External Device Recognition returned invalid device identifier. [invalid string identifier]

    To see diagnostics message you will need to subscribe to the following diagnostics topic:

    *   "diagnostics.startup.subscriptions.startupReq.topic:MIS.Device"

    However, if your Class identifies the request as coming from a device that it is not intended to undergo external device recognition, it will tell WebLogic Mobility Server that no external device recognition was performed on the request and that normal WebLogic Mobility Server device recognition should proceed. When this occurs the setDeviceID(String id) method will not be invoked when implementing the RecognizeDevice method. A diagnostics message will be logged in this scenario.

    Example: [MIS.Request.ExternalDeviceRecognition.Ignored] External Device Recognition opted not to set device identifier

    Note: To see diagnostics message you will need to subscribe to the following diagnostics topics:

    *   diagnostics.startup.subscriptions.startupReq.topic:MIS.Device

    *   diagnostics.startup.subscriptions.startupReq.level:normal

4. As a parameter to the RecognizeDevice method, you will receive the ExternalDRContext object.

    Example:

```
public class ExternalDeviceRecognizerImplTestReqParam implements
ExternalDeviceRecognizer {
    public void recognizeDevice(ExternalDeviceRecognizerContext context) {
//Implementation code here
}
}
```

5. Your implementation of the RecognizeDevice method will invoke methods of the ExternalDRContext interface via the ExternalDRContext parameter. These methods are listed in the "Public interface ExternalDRContext Methods" table below.

    Public Interface ExternalDRContext Methods

Method	Description
public String getRequestHeader(String header)	Gets the value of a request header. For example, getRequestHeader("Accept").
public String getRequestParameter(String name)	Gets the value of a request parameter. For example, getRequestParameter("Surname").
public Object getSessionAttribute(String name)	Gets the value of a session attribute. For example, getSessionAttribute("User ID").
public Cookie getRequestCookies(String name)	Gets the value of a request cookie. For example, getRequestCookies("User Login ID").
public void setDeviceID(String id) throws DeviceNotFoundException	Provides WebLogic Mobility Server with the name of the device once it has been identified. The device name must exist in the device repository

Important note: An "EntityNotFound" exception is thrown if the name of the device is not found in the Device Repository.

If the device could not be identified, a warning message is thrown: "*[MIS.Warning] External Device Recognition returned invalid device identifier [nokia66000].", and WebLogic Mobility Server device recognition will occur.

However, if you have determined that external device recognition should not be performed for this request (i.e. that normal WebLogic Mobility Server device recognition should take place) then this method should not be invoked; this will generate an information message "[MIS.Request.ExternalDeviceRecognition.Ignored] External Device Recognition opted not to set device identifier". |

6. Finally, you will need to configure the *mis.properties* file. Add the external.devicerecognition.Class attribute to the end of this file as depicted in the sample extract below:

3—Administer the Device Repository

```
##################################################
external.devicerecognition.Class: <Path and name of device recognition class>
```

Example:

```
external.devicerecognition.Class:
com.acme.devicerecognition.ExternalDeviceRecognizerImplTestReqParam
```

Note: The term "<path and name of device recognition class>" denotes the path to and name of the class that you created, as outlined in step 1 above.

7. Now, when deploying WebLogic Mobility Server to the application server, ensure that the class you created exists in the application's **WEB-INF/classes** folder.

    Note: You will need to also ensure that your code is thread-safe.

Basic Implementation Example

```java
/**
 * External Device Recognition Sample
 */

import com.mobileaware.devicerecognition.ExternalDeviceRecognizerContext;
import com.mobileaware.devicerecognition.ExternalDeviceRecognizer;
import com.mobileaware.Everix.Device.EntityNotFoundException;

public class EDRSampleCode implements ExternalDeviceRecognizer {

    public void recognizeDevice(ExternalDeviceRecognizerContext context) {

        // retrieve info from request (e.g. header)
        String phoneId = context.getRequestHeader("MSISDN");

        if (phoneId == null) {
            // in this example, having no MSISDN header present is considered an error
            // so we flag a recognition problem
            context.setDeviceId(null);
        } else {
            // perform logic e.g. lookup database for phone details based on number
            String deviceId = getDeviceIdByMSISDN(phoneId);

            if (deviceId != null) {
                try {
```

```
                              context.setDeviceId(deviceId);
                    } catch (EntityNotFoundException e) {
                              // Device identifier wasn't in the Device Repository.
                              logError("Invalid device "+deviceId+" for MSISDN "+phoneId);


                              // Default Device Recognition will continue
                    }
               } else {
                    // No device id was found for this MSISDN (perhaps it's a new subscriber).
                    // By not calling setDeviceId(), we allow default Device Recognition to continue.
               }
          }
     }

     private String getDeviceIdByMSISDN(String msisdn) { … }
     private void logError(String error) { … }
}
```

Use Device Repository Manager to Configure Device Profiles

Device Repository Manager allows you to create device profiles and, if necessary, modify existing profiles and attributes to capture more device-specific information.

Important note: Even though it is possible to do so, you should not run more than one instance of the Device Repository Manager simultaneously on a given machine. The results of doing so are undefined and may lead to data corruption and/or data loss. You should always terminate one instance before starting another.

The Device Repository File

Overview

The *DeviceRepository* file contains all of the information required to create and install the Device Repository. The file is broken into four sections

- Profile Descriptions

- Component Descriptions

- AttributeSpecs

- Devices

Sections 1 and 2 are concerned with mappings to User Agent Profile (UAProf) attributes. The AttributeSpecs section defines the attributes that can be used by each device. The devices section defines all of the known devices and their attributes.

Editing the DeviceRepository File

As it is not recommended that you edit the *DeviceRepository* file manually, you will use Device Repository Manager to add devices, remove devices and modify attribute values.

Run Device Repository Manager

Device Repository Manager can be run in Editing Mode to allow you to edit the file.

1. If you have a Windows platform, run *DeviceRepositoryManager.exe* or choose **Start → Programs → BEA WebLogic Mobility Server 3.6 → Applications → Device Repository Manager** to launch the tool.

    If you have a UNIX/Linux platform, navigate to the **applications** folder and run the Device Repository Manager application directly from there.

2. The "Device Repository options" dialog is displayed.

    Device Repository options dialog

3. Select **Edit Device File** to load the file for editing. When the file is loaded into the tool, an in-memory model of the file is created. Edits are not committed until the file is exported from the tool.

Using the Device Repository Manager Edit Device File Mode

Load the File for Editing

1. When the tool launches, you will be asked to select the file. Once you select a file, the following progress bar displays until the file loads.

    Loading File Progress Bar
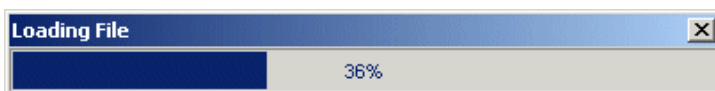
Browse the DeviceRepository file

1. The "Device Browser" is displayed with the root node selected.

   Note: The **Add Device** and **Remove Device** buttons are disabled at this stage. You cannot add or remove devices directly under the root node.

   Browse Device File
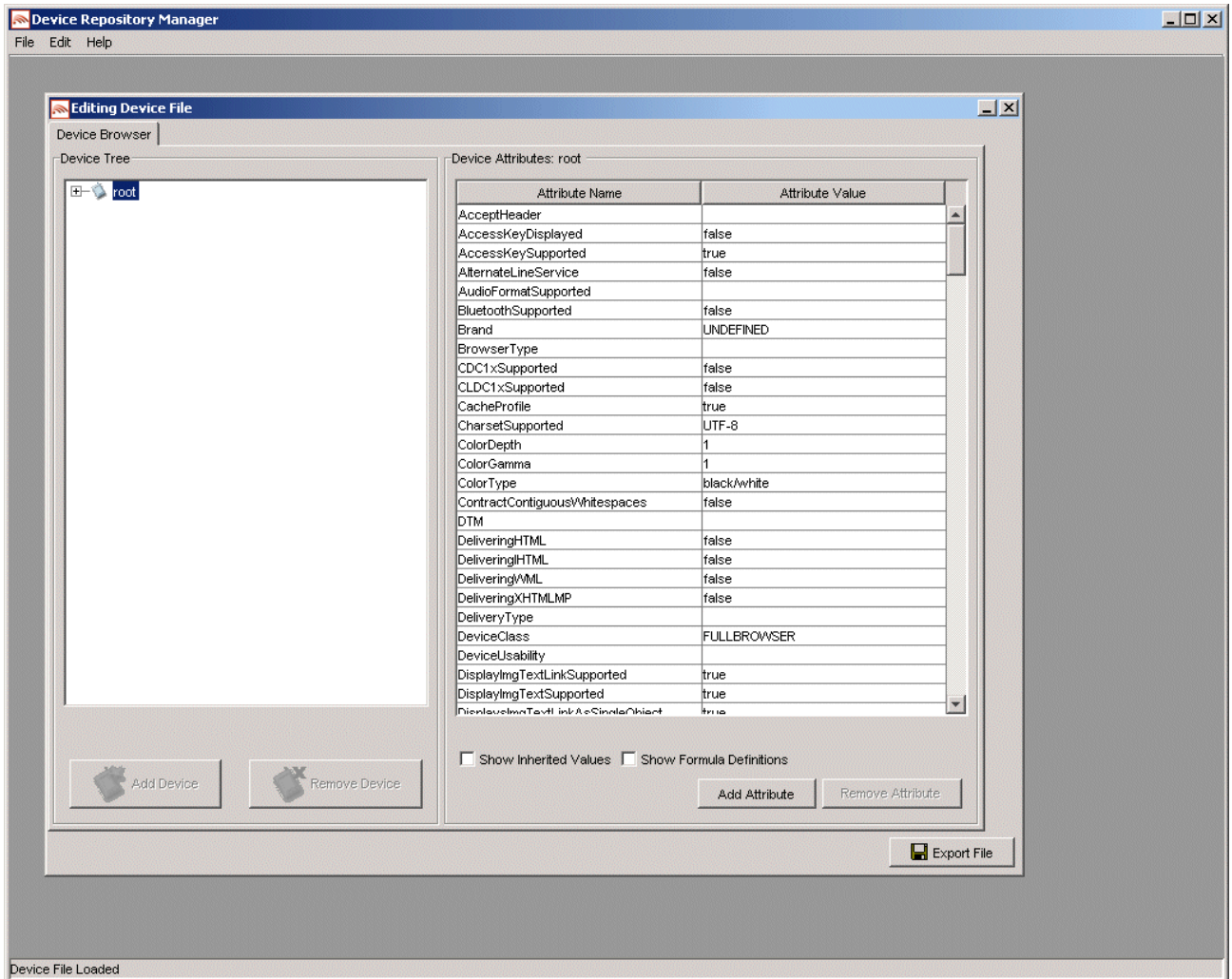
If you expand the tree and select the XHTMLMP node you will notice that the **Add Device** button becomes enabled. This indicates that devices may be added under this node.

Note: The **Remove Device** button is still disabled. Removal of non-leaf nodes is not permitted.

Expanded XHTML-MP Node
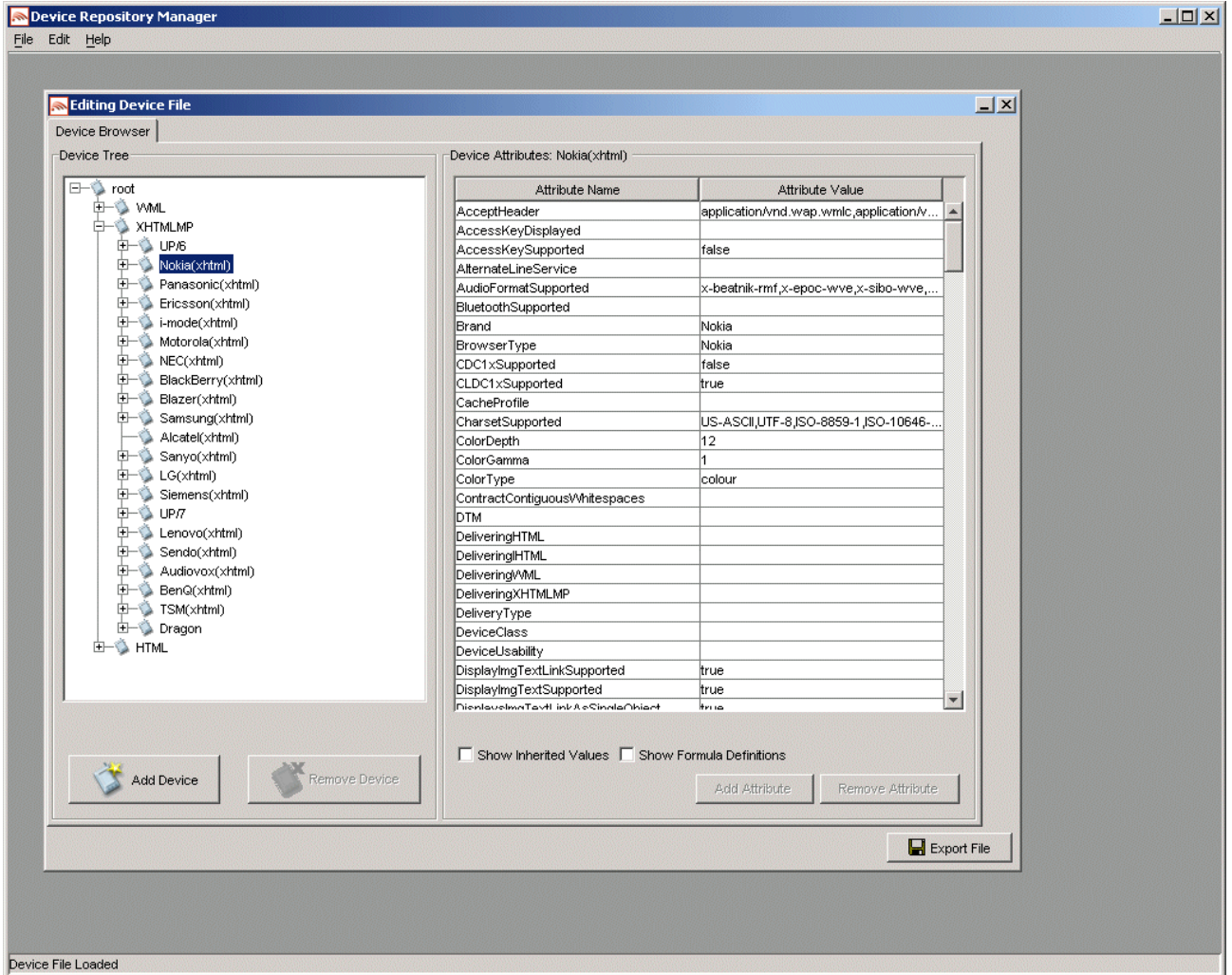
Browsing to a customer-added device enables the **Remove Device** button, indicating that the user can remove the selected node.

Note: You may only remove customer-added devices.

Customer Added Device View
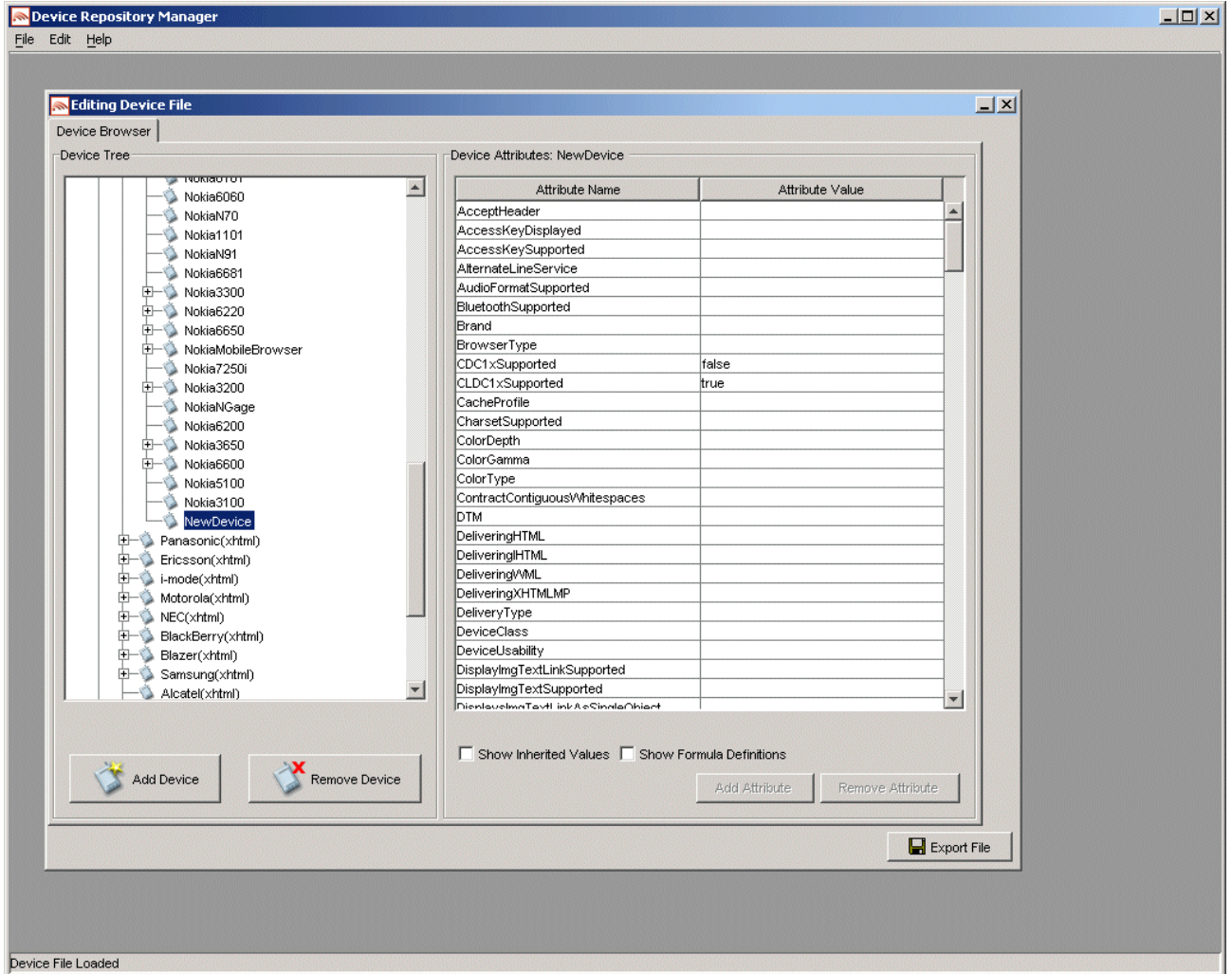
The hierarchical structure of the Device Repository allows values to be inherited from parent devices.

If you select **Show Inherited Values**, you will be able to view all of the values that are inherited from the parent devices.

Show Inherited Values View
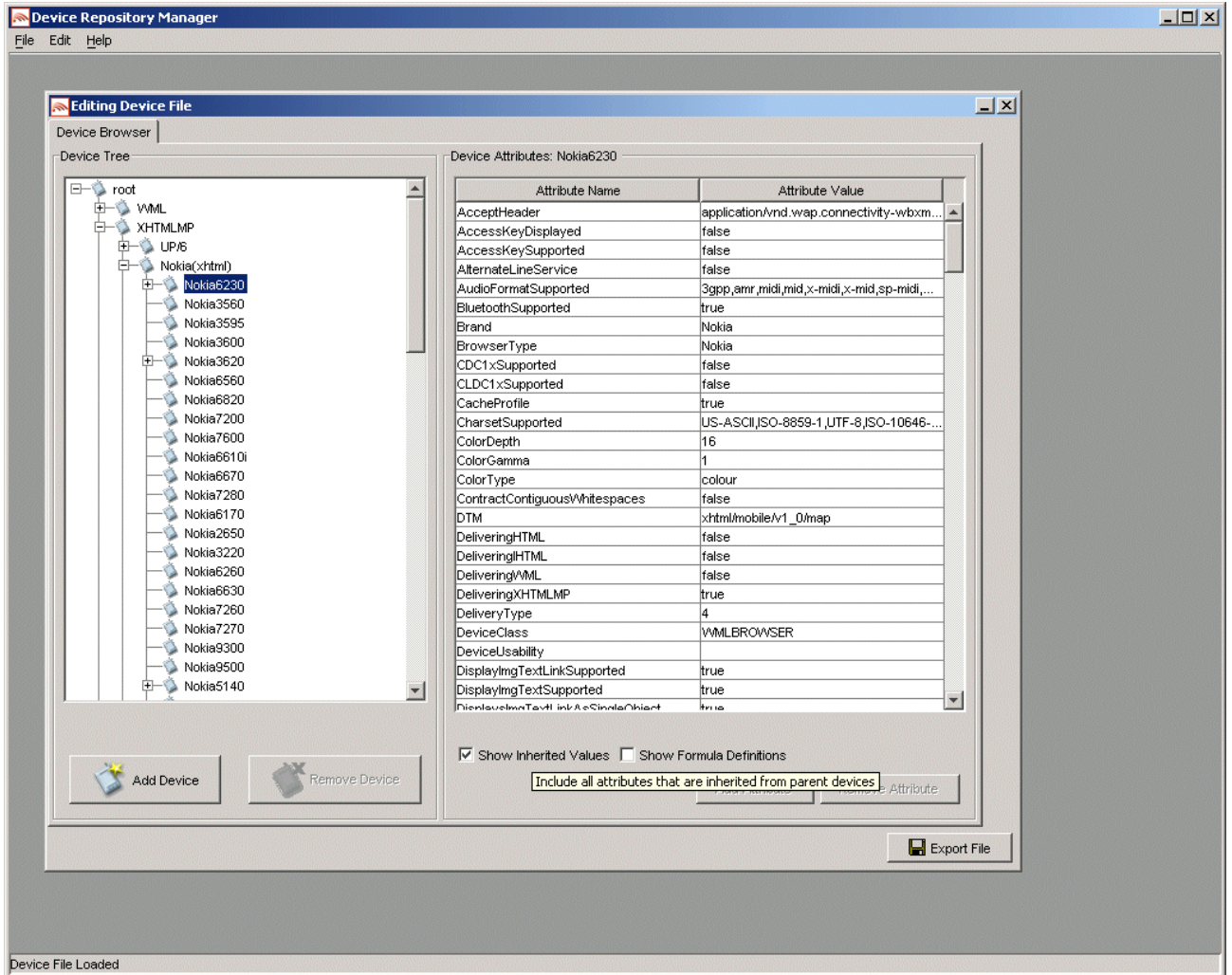
Add and Remove Custom Attributes to the DeviceRepository File

Add Attribute to the DeviceRepository File

There are three steps involved in adding a custom attribute to the *DeviceRepository* file:

- Define a name and type for the new attribute

- Set a value for the attribute

- Export the file to disk

To demonstrate this, we will use the example of adding an attribute called "SupportedImageWidth".

1. Select the **root** node in the Device Browser.
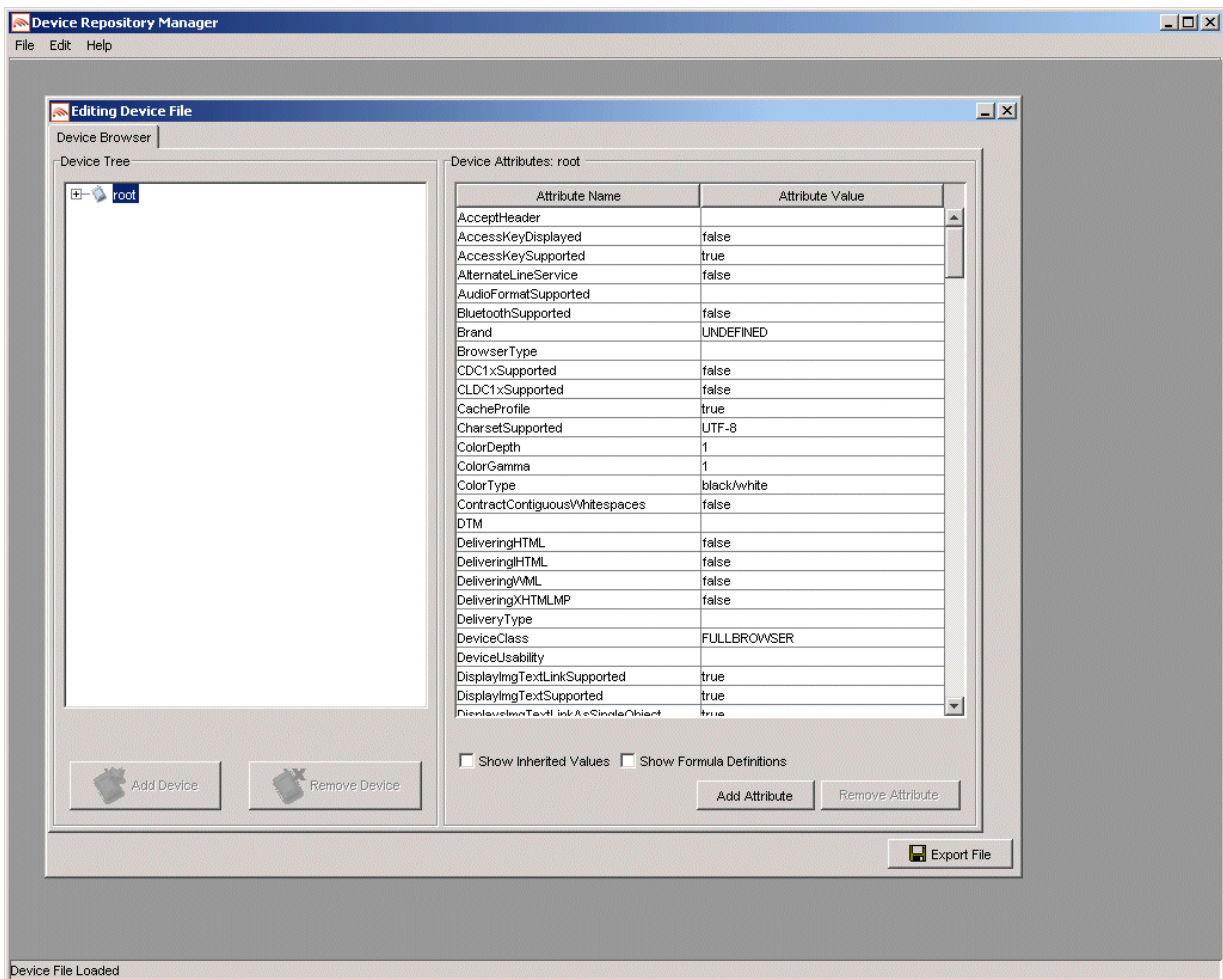
   Root node

2. There are three ways to launch the "Add New Attribute" dialog:

   - Right-click on the **root** node and choose **Add Attribute** from the menu that displays, as demonstrated in the following graphic

Add Attribute menu-option
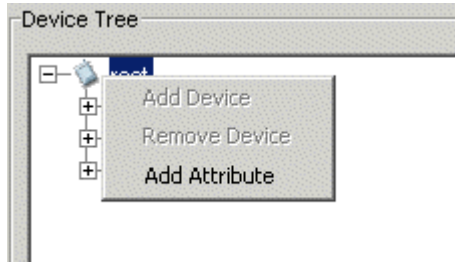
OR

- Choose **Edit → Add Custom Attribute** from the toolbar menu, as demonstrated in the following graphic

Add Custom Attribute toolbar menu-option

OR

- Click **Add Attribute** from lower right-hand-side of the browser, as illustrated.

Add Attribute button
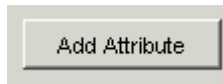
3. The "Add New Attribute" dialog will be displayed.

Add New Attribute dialog
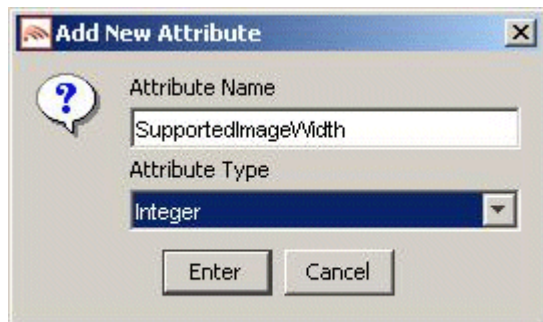
4. Enter a name for the new attribute in the **Attribute Name** field, for example, "SupportedImageWidth".

5. Specify the attribute type from the drop-down list in the **Attribute Type** field. For example, an attribute such as "SupportedImageWidth" would require a value in numeric format; therefore you would specify an attribute type of "Integer" here. Drop-down list options:

- Integer
- Boolean
- Text

Click **Enter**.

6. The following warning will be displayed.

    Warning
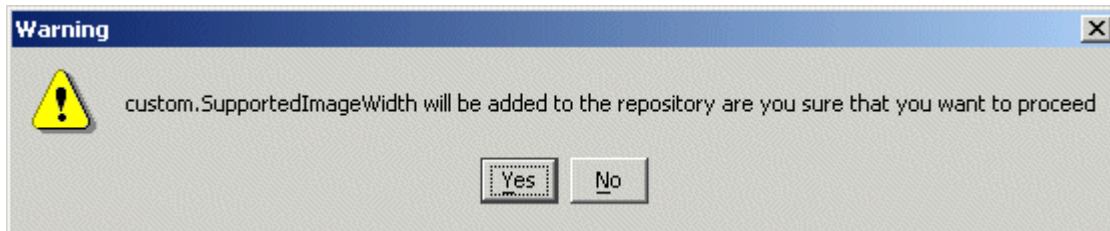
7. Click **Yes**.

8. The following message will be displayed.

    Message
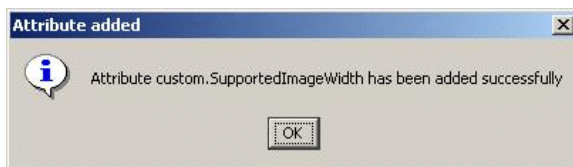
9. Click **OK**.

10. You must now set a value for the attribute.
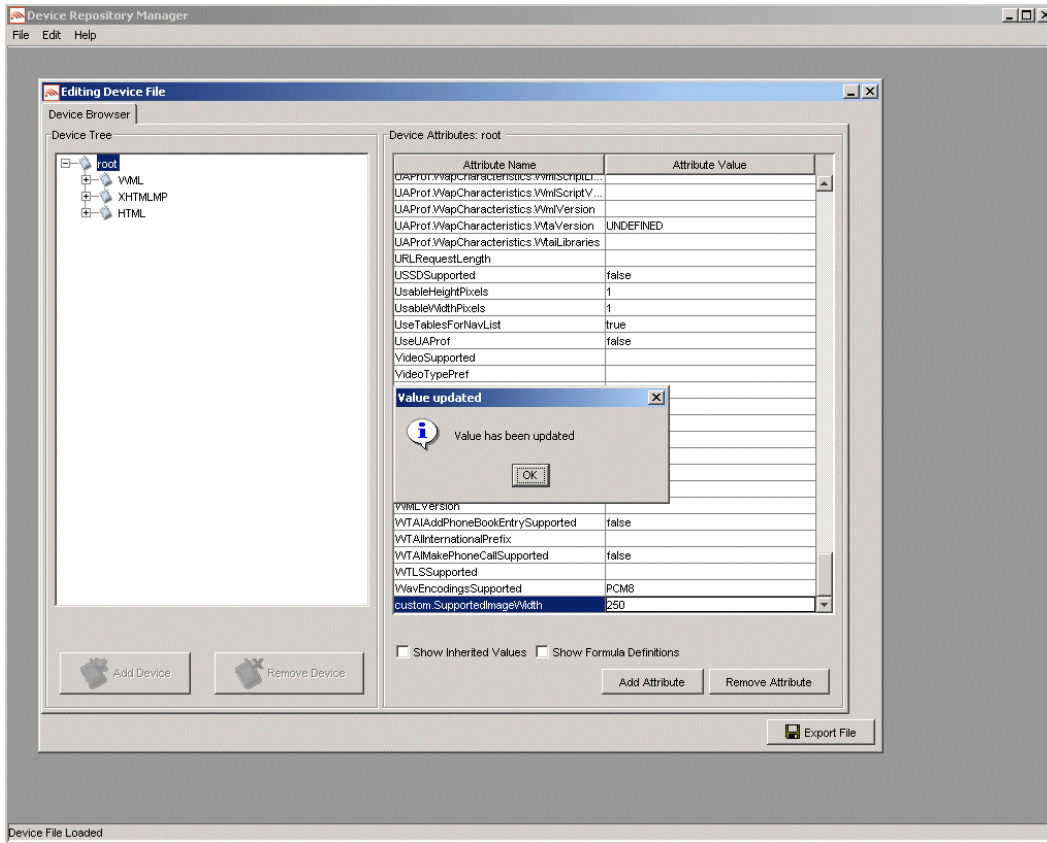
Update value

11. Locate the attribute in the **Attribute Name** column in the Device Browser window. Enter the value, for example, "250", in its corresponding field in the **Attribute Value** column and press the Enter key.

The "Value updated" message illustrated in the preceding graphic will be displayed. Click **OK**.

12. The following graphic demonstrates how to set a different value for a specific device.

Set a different value for a specific device
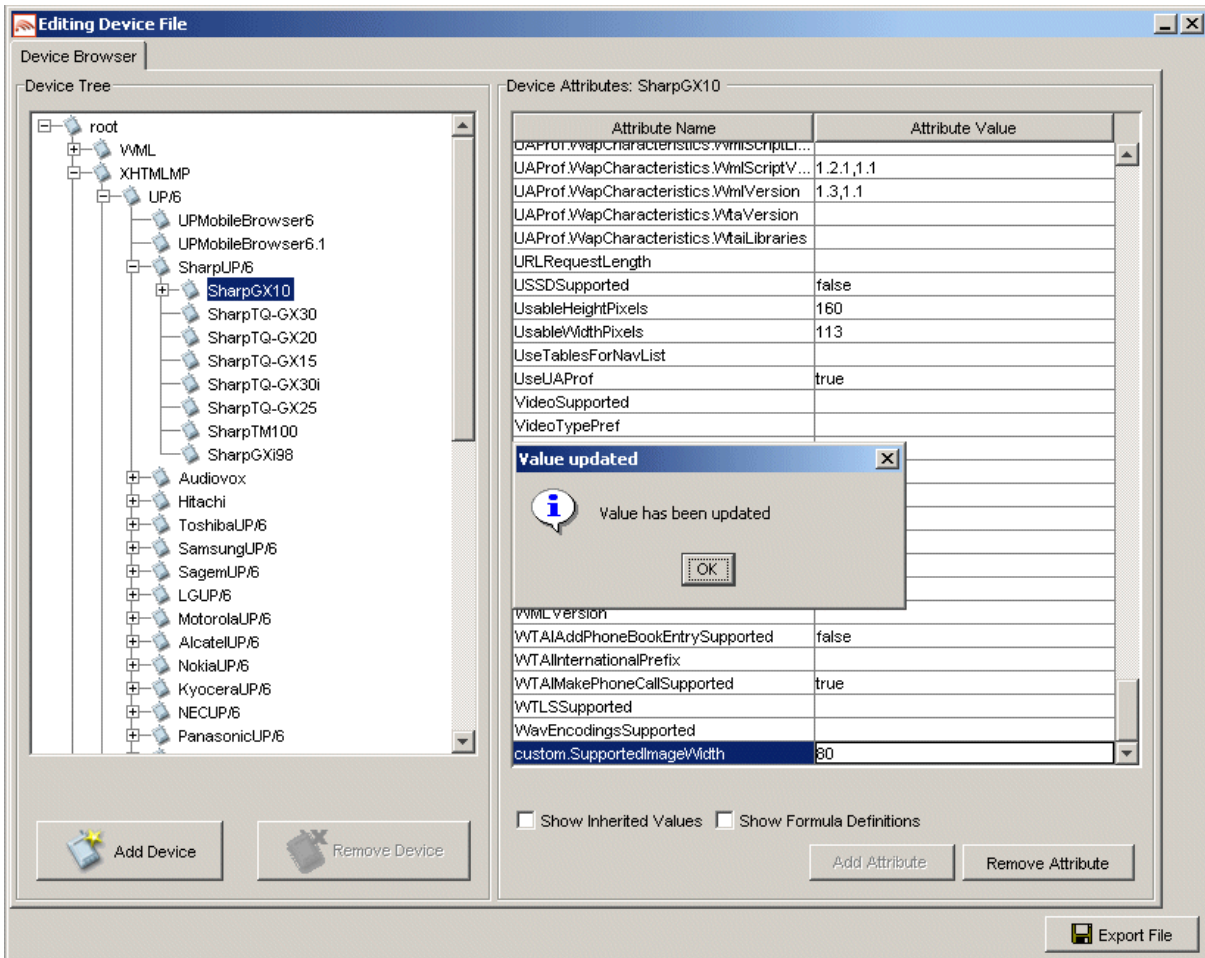
13. Navigate to the device in question in the hierarchy on the left-hand-side of the browser. Locate the attribute in the **Attribute Name** column in the Device Browser window. Enter the appropriate value in its corresponding field in the **Attribute Value** column and press the Enter key.

The "Value updated" message illustrated in the preceding graphic will be displayed. Click **OK**.

14. To successfully add the attribute to the Device Repository it is essential that you now export the file to disk.

15. Click **Export File** (from the lower right-hand-side of the browser).

16. The "Export File" dialog is displayed.

    Export File dialog
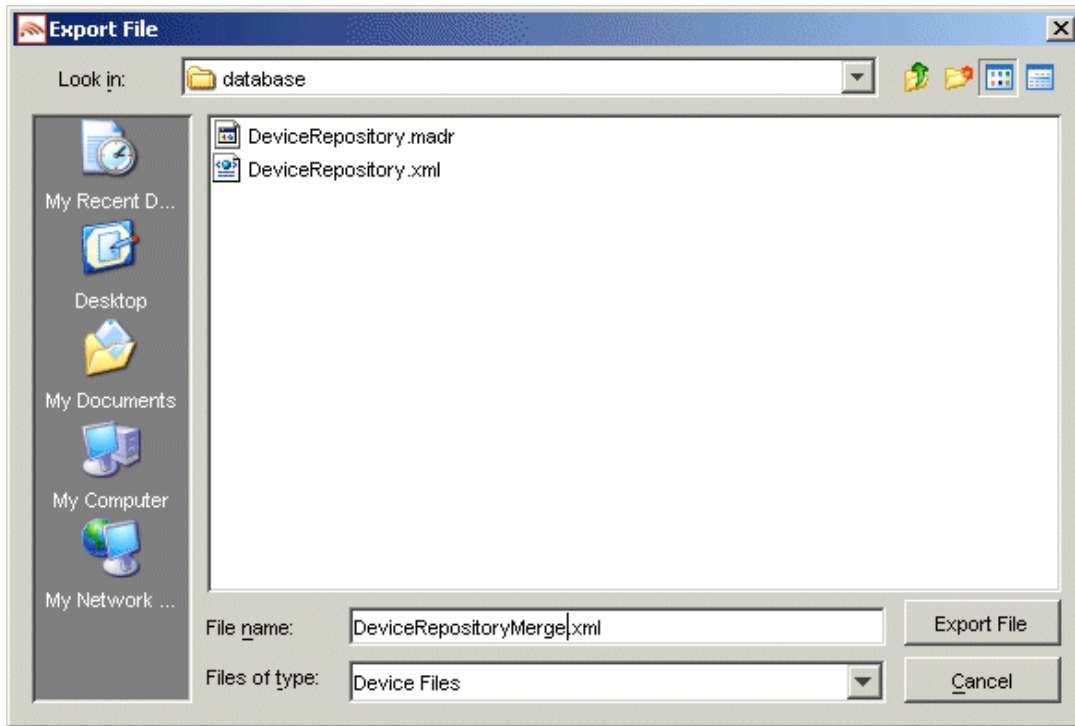
17. Specify a filename for the exported file and click **Export File**.

    Note: Files with ".madr" extensions contain compressed device repositories. If you specify a ".madr" file extension, the file that you create will be a compressed version of the Device Repository.

18. If you selected an existing file, the following message will be displayed.

    Overwrite Existing File dialog

19. Click **Yes**.

20. The file will be written to disk.

    File is written to disk
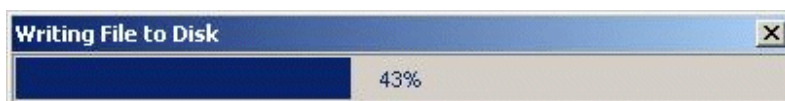
21. The following message will be displayed.

Export Complete message
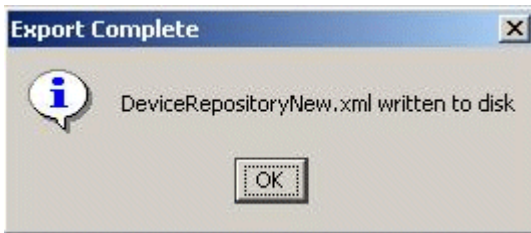
22. Click **OK**.

Remove a Custom Attribute from the File

Follow the instructions in this section to remove an attribute from the *DeviceRepository* file.

Note: It is only possible to remove *custom* attributes from the *DeviceRepository* file in this manner.

1.   Select the **root** node in the Device Browser.

   Root node
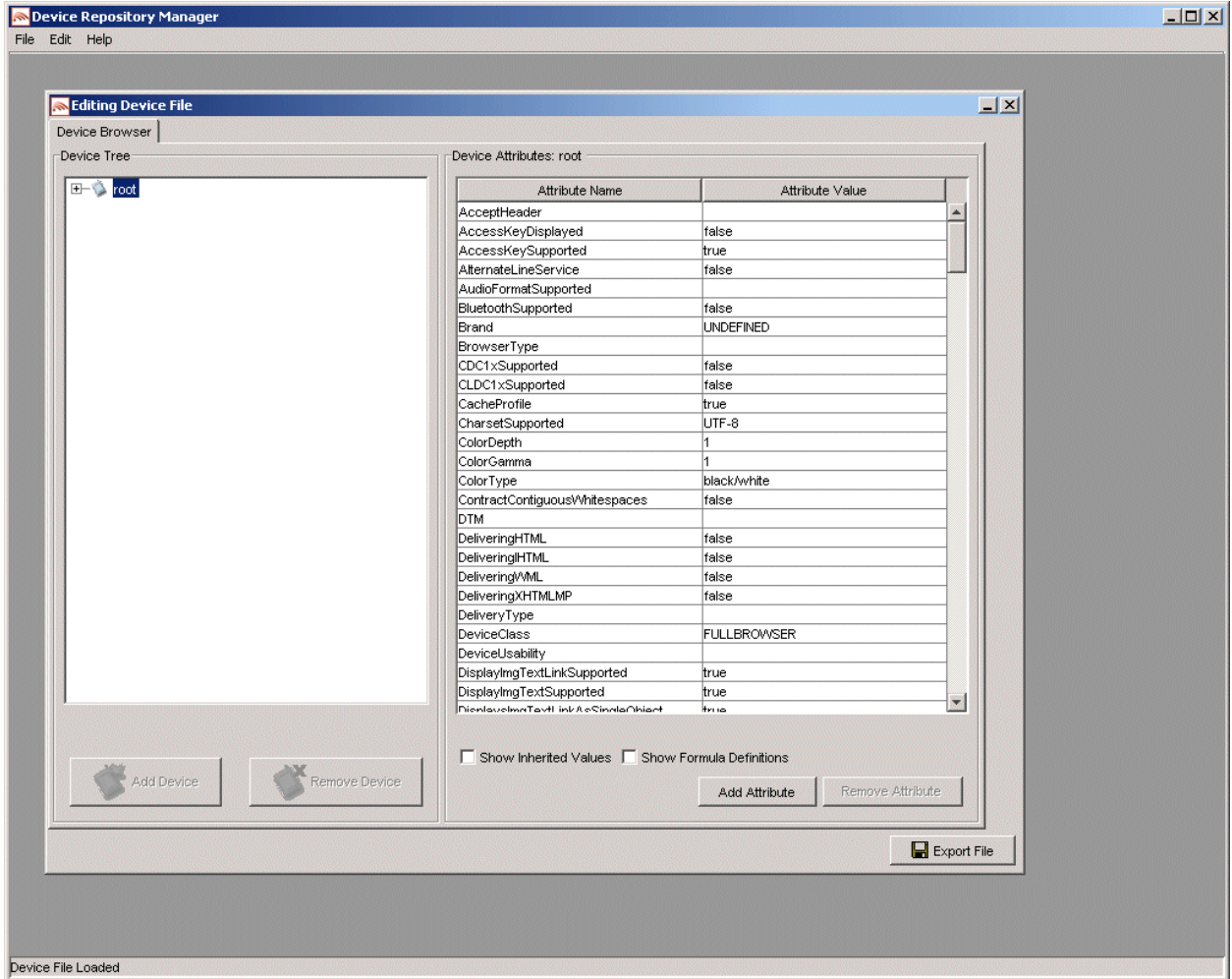
2. Select the custom attribute that you want to remove.

    Remove custom attribute
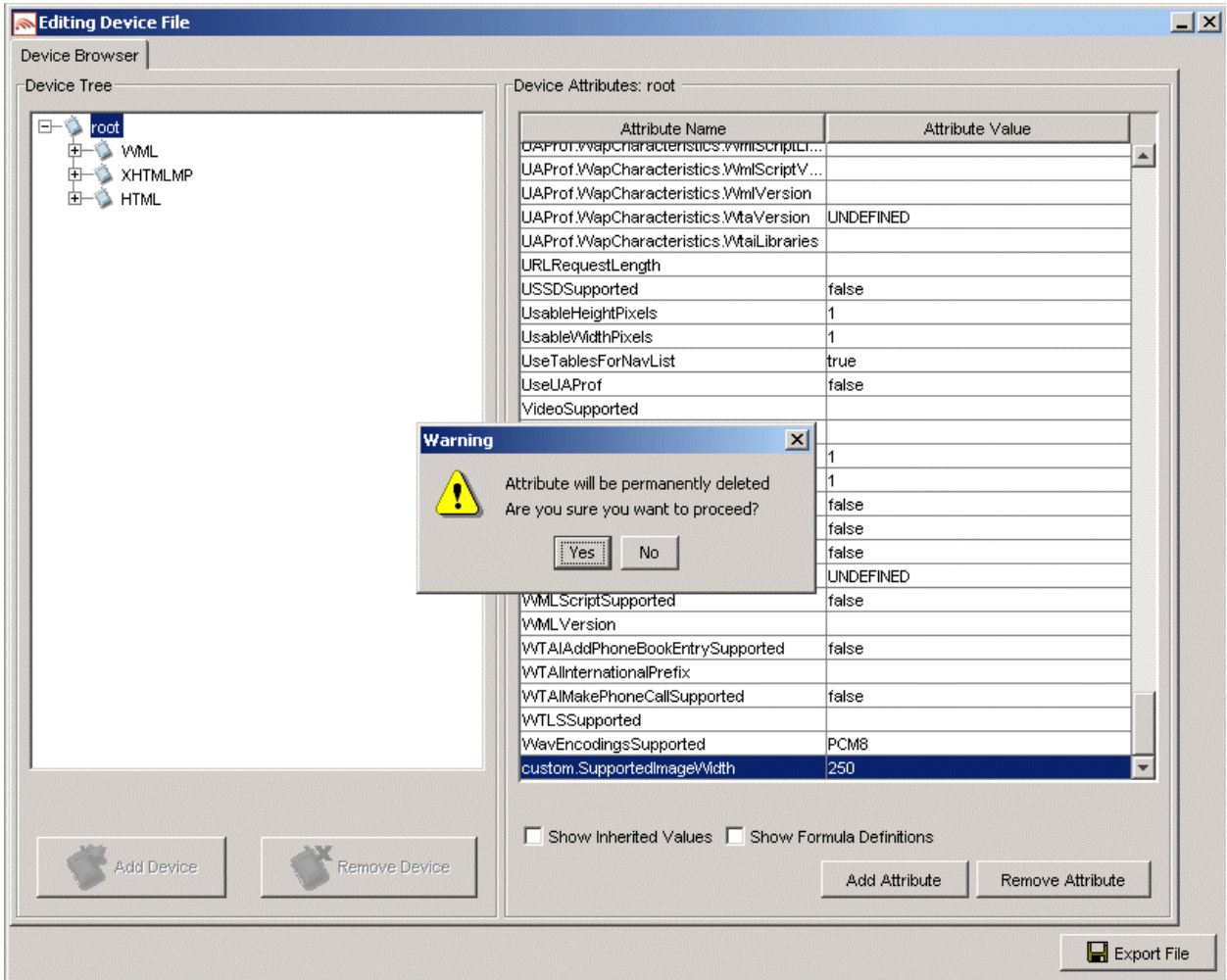
3. Click **Remove Attribute** from the lower right-hand-side of the browser.

    Note: This control is only enabled if a custom attribute is selected.

4. The "Attribute Deletion" warning message will be displayed. Click **Yes** to proceed.

5. The following message will be displayed.

    Attribute Removed message
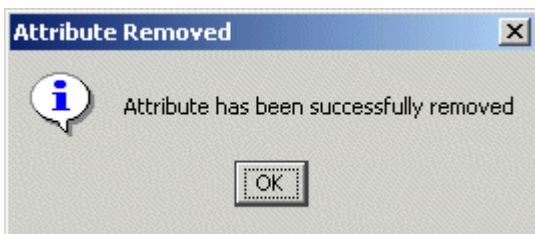
6. Click **OK**.

Add a New Device Profile

There are three steps involved in adding a new device profile:

- Complete the "Basic Details" for the device.

- Configure the standard attribute values for the device.

- Create new attributes if required.

To demonstrate this, we will use the example adding a device called Nokia9999.

Select Parent Node

You can add new devices as children of an existing device node. The parent node depends on the User-Agent string for the device and on the markup language that the phone supports.

1. The user agent for this device is "Nokia9999" and it delivers XHTML, therefore you will need to add it under the **Nokia(xhtml)** node.

   Nokia XHTML node
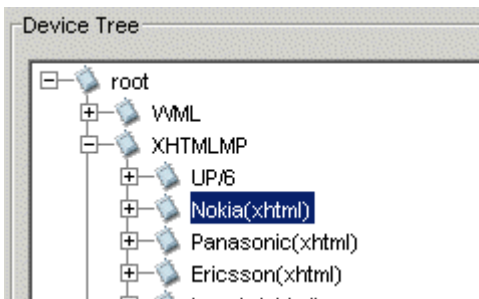
2. Select the **Nokia(xhtml)** node.

Create the New Device Node

1. Click **Add New Device**.

2. The "New Device Details" dialog is displayed.

   New Device Details Dialog

3. Enter a name and description for the device (that is, in this example, "Nokia9999") in the **Device Name** and **Description** fields, respectively.

   Note: You can specify the same values for both.

4. From the drop-down list in the **Type** field, select "device".

5. Click **OK** to create the new node.

Select the New Node

1. The attributes for the new device will display.

   Note: Only the calculated attribute values are shown. Click **Show Inherited Attributes** to display all of the attributes that are inherited from the parent devices.

Set the RecognitionUAPattern and HTTPMetaDataString Attributes

This is the most important step, where you will match the device uniquely.

1. Set the RecognitionUAPattern attribute and related attributes:

    - As the pattern matcher will firstly attempt to match values in the User-Agent header information from the incoming request with the RecognitionUAPattern attribute in the Device Repository, you will need to set this attribute first, as illustrated below. This attribute defines the regular expression or sub-string to be matched against the User-Agent header.

      Note: If the RecognitionUAPattern attribute contains a regex, then you must also set the "RecognitionRequiresRegex" attribute to true.
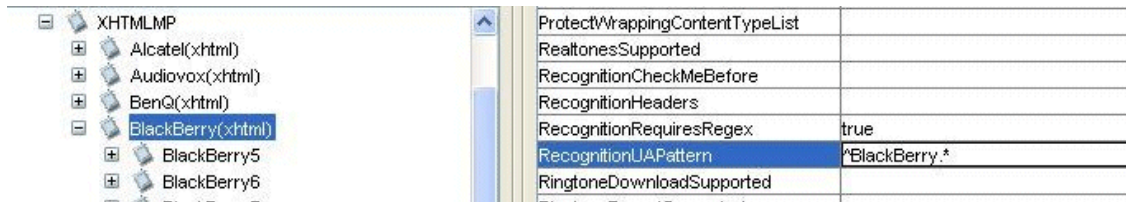
    - If there is a match here, the pattern matcher will move on to determine whether the RecognitionHeaders attribute has been set or not for specific headers in the incoming request. Therefore you will need to set the RecognitionHeaders attribute accordingly:
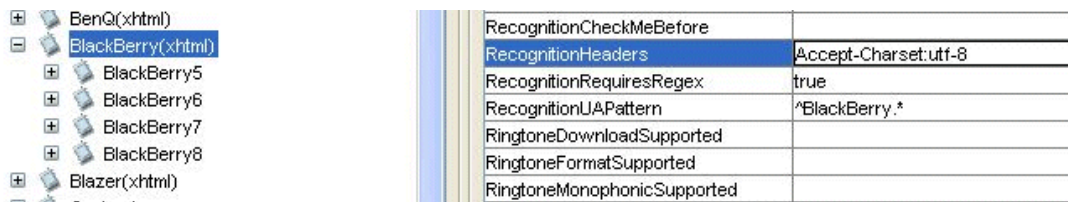
      (**Note**: If a match is NOT found, the pattern matcher will subsequently attempt to match on the HTTPMetadataKey attribute so you will need to set this—see the setting the HTTPMetadataKey attribute step below).

    - It may also be necessary to set the RecognitionCheckMeBefore attribute, to resolve potential scenarios in which a device may match more than one pattern. For example, the User-Agent "Ericsson t68i" would match the pattern "t68i", but it would also erroneously match the pattern "t68". The User-Agent "Ericsson t68" would only match the pattern "t68". Therefore, the pattern "t68i" must be checked first and if this fails to match, *then* the other pattern may be checked. The RecognitionCheckMeBefore attribute specifies a list of devices that a given device should take precedence over in the recognition process.

2. Set the HTTPMetaDataString attribute:

    The HTTPMetaDataString is a substring of the request header (typically the User-Agent header) that the phone will send that should be used for device matching.

    You must match a different substring of the User-Agent at each level in the hierarchy.

    Note: These examples assume that the commonly employed User Agent header is used here.

    Inserting the HttpMetaDataString
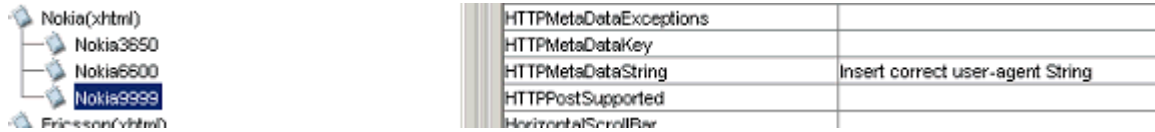
    Replace the text "Insert correct user-agent string" with the user-agent for this device. In this case, "Nokia9999" will be enough to identify the device.

    Inserting the HttpMetaDataString for the Nokia9999

    Press the Enter key to confirm the change.

Manual Update

If the correct parent node is chosen, the inherited values may be sufficient for many of the attributes. You should verify these against the device vendor's specifications and reliable third-party information websites.

It is important to populate the UAProf values first, as many of the other attributes are based on these (see "Appendix A—Device attributes").

The most important UAProf Attributes are

- UAProf.SoftwarePlatform.CcppAccept

- UAProf.HardwarePlatform.ScreenSize

- UNDEFINED

    UAProf String attributes have a default value of "UNDEFINED" on the root of the Device Repository. A value of "UNDEFINED" is inherited for an attribute in any device node if no value is defined in the manufacturers UAProf file for this device.

Setting Proprietary Device Attributes

Proprietary device attributes are either calculated or inherited from parent devices.

Note: Do not alter any attribute that is defined with isFormula="T" in the AttributeSpec.

It is recommended that you verify the following attributes:

* MaxWapDeckSize

  This attribute indicates the specific deck size, which controls the pagination of content sent to mobile devices. Typically you can set this to the same value as that of the UAProf.WapCharacteristics.WmlDeckSize attribute. However, this value may be too high for certain devices—if the UAProf value is greater than "50000", set the MaxWapDeckSize attribute to "20000".

* ImgGIFSupported

  Set this attribute to "true" if the device supports GIF files.

  Note: A device supports GIF images if "image/gif" appears in its CcppAccept attribute.

* ImgJpgBaselineSupported

  Set this attribute to "true" if the device supports JPG and JPEG files.

  Note: A device supports JPG and JPEG images if "image/jpg" or "image/jpeg" appears in its CcppAccept attribute.

* ImgPNGSupported

  Set this attribute to "true" if the device supports PNG files.

  Note: A device supports PNG images if "image/png" appears in its CcppAccept attribute.

* ImgWBMPSupported

  Set this attribute to "true" if the device supports WBMP files.

  Note: A device supports WBMP images if "image/wbmp" appears in its CcppAccept attribute.

* DTM

  This is the Device Transformation Map attribute, which enables the transformation engine to generate the correct markup for the requesting device. See the following table for details of supported DTM settings.

Supported DTM settings

Setting	Description
wml/v1_1/map	Used for WML 1.1 devices that do not support tables.
wml/v1_1/TablesSupported/map.xml	Used for WML 1.1 devices that support tables.
wml/v1_1/EricssonR380/map.xml	Used for the Ericsson R380.
wml/v1_1/UP4/SiemensSL45/map.xml	Used for the Siemens SL45.
wml/v1_1/UP4/Timeport/map.xml	Used for the Motorola Timeport.
wml/v1_2/map	Used for WML 1.2 devices.
wml/v1_3/map	Used for WML 1.3 devices.

xhtml/mobile/v1_0/map	Used for most XHTML-MP devices.
xhtml/mobile/v1_0/oma/map	Used for XHTML-MP devices that specifically require the Open Mobile Alliance defined XHTML-MP mime type settings (such as the Nokia 6600).
html/hybrid/map	Used for pseudo-PDAs such as the Blazer browser used in the Handspring Treo.
html/compact/map.xml	Used for imode devices.
html/v3_2/map.xml	Used for PDA devices that require HTML 3.2.
html/v3_2/Mozilla2/map.xml	Used for PDA Devices using the Mozilla 2 browser.
html/v3_2/IPAQ/map.xml	Used for the majority of Pocket PC based PDAs including Compaq IPAQ, HP Jornado, and so on.
html/v4_x/Mozilla4/map	Used for PC browsers.

- DeliveryType

  Ensure that this attribute is set to the same value as that of the DTM attribute. HTML4 = 1, HTML3.2 = 2, WML = 3, XHTMLMP = 4.

Appendixes

Appendix A—WebLogic Mobility Server and Device Repository Interaction

Transforming Content

Once WebLogic Mobility Server has identified a device and matched it against one in its Device Repository, it seamlessly transforms the presentation of the content to the requesting device.

This transformation is managed through the use of Device Transformation Maps (DTMs). The DTM specifies how content marked up with the WebLogic Mobility Server mobility tags is transformed to tailor the delivered page to the capabilities of the requesting device.

DTMs make it possible to accommodate new devices or upgraded versions of existing models as soon as they come on the market; transformation rules can be built quickly to take into account the new capabilities.

Each device DTM forms part of the device's profile in the Device Repository. The DTM attribute specifies the location of the transformation map to be applied to the original marked up content before it is delivered to the requesting device. The transformation map specifies how the mark-up is transformed by associating each mmXHTML/HTML tag with a Java class file that is responsible for the transformation of that tag, or by directly specifying more rudimentary transformations, such as remove or replace element.

Tailoring Content

Device profiles enable the presentation and delivery of content to be tailored to accommodate the capabilities of the requesting device.

Within WebLogic Mobility Server, tailoring of content takes place on three levels:

- When WebLogic Mobility Server identifies the requesting device, it can automatically reconfigure the presentation of content to accommodate the device's capabilities, such as splitting up a large page across a number of decks on a WAP browser.

- The content author, using the conditional mobility tags, <mm-include> and <mm-exclude>, specifies how content should be altered when being delivered to different devices. For example, the length of a product description could be tailored to accommodate different-sized screens.

- The content author creates specific layouts to target different devices or device classes. Depending on the complexity of the content, the author may choose a static layout, where the dimensions (such as the number of columns and rows in a table) are fixed. Alternatively, they may choose dynamic layouts, using the delivery context API to identify the device and using JSP methods to generate the appropriate layout "on-the-fly". For example, the author can use the API to determine the width and height of a screen, and resize the table accordingly.

Appendix B—Device Attributes

This appendix lists the current attributes in the Device Repository.

The listing is broken down into three major sections:

- CC/PP-compliant device attributes

- Proprietary device attributes

- A list of deprecated device attributes which are still supported, although their function has been replaced by a CC/PP attribute. This list will indicate which attribute should be used instead.

CC/PP-Compliant Device Attributes

The seven categories of CC/PP compliant attributes listed enable developers to create device-independent content and applications. They are listed in the Device Repository with one of the following prefixes:

1. UAProf.BrowserUA

   For more information see:
   http://wapforum.org/profiles/UAPROF/ccppschema-20020710#BrowserUA

2. UAProf.HardwarePlatform

   For more information see:
   http://wapforum.org/profiles/UAPROF/ccppschema-20020710#HardwarePlatform

3. UAProf.MmsCharacteristics

   For more information see:
   http://wapforum.org/profiles/UAPROF/ccppschema-20020710#MmsCharacteristics

4. UAProf.NetworkCharacteristics

   For more information see:
   http://wapforum.org/profiles/UAPROF/ccppschema-20020710#NetworkCharacteristics

5. UAProf PushCharacteristics

   For more information see:
   http://wapforum.org/profiles/UAPROF/ccppschema-20020710#PushCharacteristics

6. UAProf.SoftwarePlatform

   For more information, see:
   http://wapforum.org/profiles/UAPROF/ccppschema-20020710#SoftwarePlatform

7. UAProf.WapCharacteristics

   For more information:
   http://wapforum.org/profiles/UAPROF/ccppschema-20020710#WapCharacteristics

CC/PP Device Attributes – UAProf.BrowserUA Prefix

Attribute	Data Type	Example	Description
BrowserName	Literal	"Mozilla", "MSIE", "WAP42"	Name of the browser user agent associated with the current request.
BrowserVersion	Literal	"1.0"	Version of the browser.
DownloadableBrowserApps	Literal (bag)	"application/x-java-vm/java-applet"	List of executable content types which the browser supports and which it is to accept from the network. The property value is a list of MIME types, where each item in the list is a content type descriptor as specified by RFC 2045.
FramesCapable	Boolean	true | false	Set to "true" if the device browser is capable of displaying frames.
HtmlVersion	Literal	"2.0", "3.2", "4.0"	Version of HyperText Markup Language (HTML) supported by the browser.
JavaAppletEnabled	Boolean	true | false	Set to "true" if the device browser supports Java applets.
JavaScriptEnabled	Boolean	true | false	Set to "true" if the device browser supports JavaScript.
JavaScriptVersion	Literal	"1.4"	Version of the JavaScript language supported by the browser.
PreferenceForFrames	Boolean	true | false	Set to "true" if the user's preference is to receive HTML content that contains frames.
TablesCapable	Boolean	true | false	Set to "true" if the device browser is capable of displaying tables.
XhtmlVersion	Literal	"1.0"	Version of XHTML supported by the browser.
XhtmlModules	Literal (bag)	"XHTML1-struct", "XHTML1-blkstruct", "XHTML1-frames"	List of XHTML modules supported by the browser. Property value is a list of module names, where each item in the list is the name of an XHTML module as defined by the W3C document "Modularization of XHTML", Section 4. List items are separated by white space. Note that the referenced document is a work in progress. Any subsequent changes to the module naming conventions should be reflected in the values of this property.

CC/PP Device Attributes – UAProf.HardwarePlatform Prefix

Attribute	Data Type	Example	Description
BluetoothProfile	Literal (bag)	"dialup", "lanAccess"	Supported Bluetooth profiles as defined in the Bluetooth specification [BLT].
BitsPerPixel	Number (integer)	"2", "8"	The number of bits of color or grayscale information per pixel, related to the number of colors or shades of gray the device can display.
ColorCapable	Boolean	true | false	Set to "true" if the device's display supports color. "true" means color is supported. "false" means the display supports only grayscale or black and white. Type: Boolean Resolution.
CPU	Literal (string)	"Pentium III", "PowerPC 750"	Name and model number of the device CPU.
ImageCapable	Boolean	true | false	Set to "true" if the device supports the display of images. If the value is "true", the property CcppAccept may list the types of images supported.
InputCharSet	Literal (bag)	"US-ASCII", "ISO-8859-1", "Shift_JIS"	List of character sets supported by the device for text entry. Property's value is a list of character sets, where each item in the list is a character set name, as registered with IANA.
Keyboard	Literal (string)	"Disambiguating", "Qwerty", "PhoneKeypad"	Type of keyboard supported by the device, as an indicator of ease of text entry.
Model	Literal (string)	"K800i", "Q30"	Model number assigned to the device by the vendor or manufacturer
NumberOfSoftKeys	Number (integer)	"3", "2"	Number of soft keys available on the device.
OutputCharSet	Literal (bag)	"US-ASCII", "ISO-8859-1", "Shift_JIS"	List of character sets supported by the device for output to the display. Property value is a list of character sets, where each item in the list is a character set name, as registered with IANA.
PixelAspectRatio	Dimension (pair of numbers)	"1x2"	Ratio of pixel width to pixel height.

PointingResolution	Literal (string)	"Character", "Line", "Pixel"	Type of resolution of the pointing accessory supported by the device.
ScreenSize	Dimension (pair of numbers)	"160x160", "640x480"	The size of the device's screen in units of pixels, composed of the screen width and the screen height.
ScreenSizeChar	Dimension	"12x4", "16x8"	Size of the device's screen in units of characters, composed of the screen width and screen height. The device's standard font should be used to determine this property's value. (Number of characters per row)x(Number of rows). In calculating this attribute use the largest character in the device's default font.
StandardFontProportional	Boolean	true | false	Set to "true" if the device's standard font is proportional.
SoundOutputCapable	Boolean	true | false	Set to "true" if the device supports sound output through an external speaker, headphone jack, or other sound output mechanism.
TextinputCapable	Boolean	true | false	Set to "true" if the device supports alpha-numeric text entry. "true" means the device supports entry of both letters and digits. "false" means the device supports only entry of digits.
Vendor	Literal	"Nokia"	Name of the vendor manufacturing the device.
VoiceinputCapable	Boolean	true | false	Set to "true" if the device supports any form of voice input, including speech recognition. This includes voice-enabled browsers.

CC/PP Device Attributes – UAProf.MmsCharacteristics Prefix

Attribute	Data Type	Example	Description
MmsCcppAccept	Bag	"text/html"	List of content types the device supports, which can be carried inside an MMS message.
MmsCcppAcceptCharSet	Bag	"US-ASCII"	The accepted character set.
MmsMaxImageResolution	String	"120x160"	The maximum image resolution supported by the device for MMS messages.

MmsMaxMessageSize	Integer	"1397"	The maximum size of an MMS message supported by the device.
MmsVersion	Bag	"1.0"	The version of MMS supported by the device.

CC/PP Device Attributes – UAProf.NetworkCharacteristics Prefix

Attribute	Data Type	Example	Description
SupportedBluetoothVersion	Literal	"1.0"	Supported Bluetooth version.
CurrentBearerService	Literal	"OneWaySMS", "GUTS", "TwoWayPacket"	The bearer on which the current session was opened.
SecuritySupport	Literal (bag)	"WTLS-1", WTLS-2", "WTLS-3", "signText", "PPTP"	List of types of security or encryption mechanisms supported by the device.
SupportedBearers	Literal (bag)	"GPRS", "GUTS", "SMS", CSD", "USSD"	List of bearers supported by the device.

CC/PP Device Attributes – UAProf.PushCharacteristics Prefix

Attribute	Data Type	Example	Description
Push-Accept	Literal (bag)	"text/html", "text/plain", "image/gif"	List of content types the device supports that can be carried inside the message/http entity body when OTA-HTTP is used. Property value is a list of MIME types, where each item in the list is a content type descriptor as specified by RFC 2045.
Push-Accept-Charset	Literal (bag)	"US-ASCII", "ISO-8859-1", "Shift_JIS"	List of character sets the device supports. Property value is a list of character sets, where each item in the list is a character set name registered with IANA.
Push-Accept-Encoding	Literal (bag)	"base64", "quoted-printable"	List of transfer encodings the device supports. Property value is a list of transfer encodings, where each item in the list is a transfer encoding name as specified by RFC 2045 and registered with IANA.
Push-Accept-Language	Literal (sequence)	"zh-CN", "en", "fr"	List of preferred document languages. If a resource is available in more than one natural language, the server can use this

			property to determine which version of the resource to send to the device. The first item in the list should be considered the user's first choice, the second the second choice, and so on. Property value is a list of natural languages, where each item in the list is the name of a language as defined by RFC 3066.
Push-Accept-AppID	Literal (bag)	"x-wap-application:wml.ua", "*"	List of applications the device supports, where each item in the list is an application-id on absoluteURI format as specified in [PushMsg]. A wildcard ("*") may be used to indicate support for any application.
Push-MsgSize	Number	"1024", "1400"	Maximum size of a push message that the device can handle. Value is number of bytes.
Push-MaxPushReq	Number	"1", "5"	Maximum number of outstanding push requests that the device can handle.

CC/PP Device Attributes – UAProf.SoftwarePlatform Prefix

Attribute	Data Type	Example	Description
AcceptDownloadableSoftware	Boolean	true | false	Set to "true" if the user's preference is to accept downloadable software.
AudioinputEncoder	Literal (bag)	"G.711"	List of audio input encoders supported by the device
CcppAccept	Literal (bag)	"text/html", "text/plain", "text/html", "image/gif"	List of content types the device supports. Property value is a list of MIME types, where each item in the list is a content type descriptor as specified by RFC 2045.
CcppAccept-Charset	Literal (bag)	"US-ASCII", "ISO-8859-1", "Shift_JIS"	List of character sets the device supports. Property value is a list of character sets, where each item in the list is a character set name registered with IANA.
CcppAccept-Encoding	Literal (bag)	"base64", "quoted-printable"	List of transfer encodings the device supports. Property value is a list of transfer encodings, where each item in the list is a transfer encoding name as specified by RFC 2045 and registered with IANA.
CcppAccept-Language	Literal	"zh-CN", "en", "fr"	List of preferred document languages. If

	(sequence)		a resource is available in more than one natural language, the server can use this property to determine which version of the resource to send to the device. The first item in the list should be considered the user's first choice, the second the second choice, and so on. Property value is a list of natural languages, where each item in the list is the name of a language as defined by RFC 3066[RFC3066].
DownloadableSoftwareSupport	Literal (bag)	"application/x-msdos-exe"	List of executable content types which the device supports and which it is willing to accept from the network. The property value is a list of MIME types, where each item in the list is a content type descriptor as specified by RFC 2045.
JavaEnabled	Boolean	true | false	Set to "true" if the device supports a Java virtual machine.
JavaPlatform	Literal (bag)	"Pjava/1.1.3-compatible", "MIDP/1.0-compatible", "J2SE/1.0-compatible"	The list of Java platforms and profiles installed in the device. Each item in the list is a name token describing compatibility with the name and version of the java platform specification or the name and version of the profile specification name (if profile is included in the device).
JVMVersion	Literal (bag)	"SunJRE/1.2", "MSJVM/1.0"	List of the Java virtual machines installed on the device. Each item in the list is a name token describing the vendor and version of the VM.
MexeClassmarks	Literal (bag)	"1", "3"	List of MExE classmarks supported by the device. Value "1" means the MExE device supports WAP. Value "2" means MExE device supports Personal Java, value "3" means that MExE device supports MIDP applications and value "4" means the device supports the CLI Platform. All other values should be considered reserved for use by MexE.
MexeSpec	Literal	"7.02"	Class mark specialization. Refers to the first two digits of the version of the MExE Stage 2 spec.
MexeSecureDomains	Boolean	true | false	Set to "true" if the device supports MExE security domains. "true", means that security domains are supported in accordance with MExE specifications

			identified by the MexeSpec attribute. "false" means that security domains are not supported and that the device does not have a trusted domain (area).
OSName	Literal	"Mac OS", "Windows NT"	Name of the device's operating system.
OSVendor	Literal	"Apple", "Microsoft"	Vendor of the device's operating system
OSVersion	Literal	"6.0", "4.5"	Version of the device's operating system.
RecipientAppAgent	Literal	"BrowserMail"	User agent associated with the current request. Value should match the name of one of the components in the profile. A component name is specified by the ID attribute on the prf:Component element containing the properties of that component..
SoftwareNumber	Literal	"2"	Version of the device-specific software (firmware) to which the device's low-level software conforms.
VideoinputEncoder	Literal (bag)	"MPEG-1", "MPEG-2", "H.261"	List of video input encoders supported by the device.
Email-URI-Schemes	Literal (bag)	"pop", "imap", "http", "https"	List of URI schemes the device supports for accessing e-mail. Property value is a list of URI schemes, where each item in the list is a URI scheme as defined in RFC 2396.
JavaPackage	Literal (bag)	"com.acme.regexp/1.1", "com.acme.helper/3.0"	(From J2EE Client Provisioning) Details about optional packages installed on the device over and above those that are part of the Java profile, and the versions of these additional packages.
JavaProtocol	Literal (bag)	"SMS/1.0", "FILE/1.0"	(from J2EE Client Provisioning) Details about protocols supported by the device over and above those that are part of the standard Java profile indicated and the versions of these additional protocols.
CLIPlatform	Literal (bag)	"Standard CLI 2002/Compact", "Standard CLI 2002/Kernel"	The list of standard Common Language Infrastructure platforms and profiles installed in the device. Each item in the list is a name token describing the name and edition of the CLI platform specification including the name of the profile specification.

CC/PP Device Attributes – UAProf.WapCharacteristics prefix

Attribute	Data Type	Example	Description
SupportedPictogramSet	Literal (bag)	"core", "core/operation", "human"	Pictogram classes supported by the device as defined in "WAP Pictogram specification".
WapDeviceClass	Literal	"A"	Classification of the device based on capabilities as identified in the WAP 1.1 specifications. Current values are "A", "B" and "C".
WapVersion	Literal	"1.1", "1.2.1", "2.0"	Version of WAP supported.
WmlDeckSize	Number	"4096"	Maximum size of a WML deck that can be downloaded to the device. This may be an estimate of the maximum size if the true maximum size is not known. Value is number of bytes.
WmlScriptLibraries	Literal (bag)	"LANG", "FLOAT", "STRING", "URL", "WMLBROWSER", "DIALOGS", "PSTOR"	List of mandatory and optional libraries supported in the device's WMLScript VM.
WmlScriptVersion	Literal (bag)	"1.1", "1.2"	List of WMLScript versions supported by the device. Property value is a list of version numbers, where each item in the list is a version string conforming to Version.
WmlVersion	Literal (bag)	"1.1", "2.0"	List of WML language versions supported by the device. Property value is a list of version numbers, where each item in the list is a version string conforming to Version.
WtaiLibraries	Literal (bag)	"WTAVoiceCall", "WTANetText", "WTAPhoneBook", "WTACallLog", "WTAMisc", "WTAGSM", "WTAIS136", "WTAPDC"	List of WTAI network common and network specific libraries supported by the device. Property value is a list of WTA library names, where each item in the list is a library name as specified by "WAP WTAI" and its addendums. Any future addendums to "WAP WTAI" should be reflected in the values of this property.
WtaVersion	Literal	"1.1"	Version of WTA user agent.
DrmClass	Literal (bag)	"ForwardLock", "CombinedDelivery",	DRM Conformance Class as defined in OMA-Download-DRM-v1_0.

		"SeparateDelivery"	
DrmConstraints	Literal (bag)	"datetime", "interval"	DRM permission constraints as defined in OMA-Download-DRMREL-v1_0. The datetime and interval constraints depend on having a secure clock in the terminal.
OmaDownload	Boolean	true | false	Set to "true" if the device supports OMA Download as defined in OMA-Download-OTA-v1_0.

Proprietary Device Attributes

These attributes are a more robust set of device characteristics describing device characteristics that are not currently covered by the CC/PP standards. They can be used to further fine-tune web content and applications.

Proprietary Device Attributes

Proprietary Attribute	Data Type	Example Values	Description
AccessKeyDisplayed	Boolean	true | false	Set to "true" if the browser displays the number assigned to access key beside the relevant link.
AccessKeySupported	Boolean	true | false	Set to "true" if the browser supports access keys.
AlternateLineService	Boolean	True | false	Set to "true" if a device can make a voice call while keeping a data call online.
BluetoothSupported	Boolean	true | flse	Set to "true" if the device is Bluetooth enabled.
Brand	String	Nokia	Name of the device manufacturer.
BrowserType	String	Openwave	Name of the browser.
ColorGamma	Integer	1	The color gamma of the device.
ContractContiguousWhitespaces	Boolean	true | false	Set to "true" if the device does not contract insignificant white space when rendering markup.
DTM	String	Path to the DTM	Indicates the relevant transformation map for a device.

DeliveringHTML	Boolean	true | false	Set to "true" if the product will deliver HTML to a given device. Can be used to target content at HTML devices.
DeliveringIHTML	Boolean	true | false	Set to "true" if the product will deliver IHTML to a given device. Can be used to target content and imode devices.
DeliveringWML	Boolean	true | false	Set to "true" if the product will deliver WML to a given device. Can be used to target content at WML devices.
DeliveringXHTMLMP	Boolean	true | false	Set to "true" if the product will deliver XHTML MP to a given device. Can be used to target content at XHTML-MP devices.
DeliverTableborder	Boolean	true | false	If set to "true", this device supports the border attribute on the table element. Where present in the source, this attribute should be sent to the device.
DeliverTableCellpadding	Boolean	true | false	If set to "true", this device supports the cellpadding attribute on the table element. Where present in the source, this attribute should be sent to the device.
DeliverTableCellspacing	Boolean	true | false	If set to "true", this device supports the cellspacing attribute on the table element. Where present in the source, this attribute should be sent to the device.
DeliveryType	Integer	1 or 2 or 3 or 4 where: 1 = HTML 2 = WindowsCE 3 = WML 4 = XHTML MP	Specifies the type of content that can be sent to the device.
DeviceUsability	String	DeviceUsability_MEDIUM	Describes the usability of the devices user interface.
DisplayImgTextlinkSupported	Boolean	True | false	Set to "true" if images, text, and links can be rendered on

			the same line on the browser.
DisplayImgTextSupported	Boolean	true | false	Set to "true" if images and text can be rendered on the same line on the browser.
DisplaysImgTextlinkAsSingleObject	Boolean	true | false	Set to "true" if the device renders a <img><a href…> as a single object.
DisplaysMultipleImagesOnSameLine	Boolean	true | false	Set to "true" if the device supports multiple images on the same line.
DisplaysWMLSelectAsNumberedList	Boolean	true | false	Set to "true" if the device renders a WML Select List as a numbered list.
DownloadFunSupported	Boolean	True | False	Set to "true" if Openwave Download Fun objects can be sent to the device.
EMSSupported	Boolean	True | false	Set to "true" if the device supports EMS.
TransformCSS	Boolean	True | False	Set to "true" if the product will apply CSS on the server-side for this device.
FallbackRecognitionLogic	String	E.G: Accept:"wml" | x-wap-profile-diff:"") & !(User-Agent:"mozilla" | User-Agent:"Mozilla" "None"—When FallbackRecognitionLogic is not set, a value of "None" should be used.	Any existing values here will override the HTTPMetaDataKey and HTTPMetaDataString attributes to allow a more advanced mechanism for determining whether or not a node should be matched during device recognition. The attribute allows multiple headers to be considered during the recognition process. IMPORTANT NOTE: Like any other attribute, the FallbackRecognitionLogic expression will be inherited by child nodes, which is unlikely to be the intended behaviour. Therefore if child nodes do not have their own recognition logic expression, they should be given the special value of "none" for this attribute. In

			particular, all direct children of the WML and XHTMLMP nodes should initially be given a FallbackRecognitionLogic value of "None". For more information, see the "Appendix E—Fallback Recognition Logic Expression Language Details" section.
FlashSupported	Boolean	true | false	Set to "true" if the device supports Flash.
FormSelectRenderedAsDropDown	Boolean	true | false	Set to "true" if the form <select> element is rendered as a drop down list.
FormSelectRenderedAslink	Boolean	true | false	Set to "true" if the form <select> element is rendered as a link to another card where the user makes the selection.
FormSelectRenderedAsList	Boolean	true | false	Set to "true" if the form <select> element is rendered as a list, with all options displayed.
ForwardLockContentTypeList	String	application/vnd.oma. drm.message	Indicates the content types supported for DRM Forward Lock.
HTTPMetaDataExceptions	String	Opera, Mozilla/5, and so on.	Indicates HTTPMetaDataStrings that should NOT be considered a match during device matching. Some User Agent strings contain generic values that can could potential cause a false match to occur. Filling in this field will allow device matching to progress further down the device hierarchy.
HTTPMetaDataKey	String	User-Agent Accept UA-OS	Indicates which part of the device's header contains the device's unique signature.
HTTPMetaDataString	String	Nokia6210	Device's unique header string.

HTTPPostSupported	Boolean	true | false	Set to "true" if the device supports the HTTP post method.
HorizontalScrollBar	Boolean	true | false	Set to "true" if device supports a horizontal scroll bar.
IRDASupported	Boolean	true | false	Set to "true" if the device supports Infrared Data Association standards for wireless transfer of data from one device to another.
ImagesPlacedOnNewline	Boolean	true | False	Set to "true" if the device places images on a new line.
ImgAslinkSupported	Boolean	true | false	Set to "true" if the browser can render an image in <a href> tags as a hyperlink.
ImgGIFSupported	Boolean	True | False	Set to "true" if the browser supports GIF images.
ImgGifAnimatedSupported	Boolean	True | False	Set to "true" if the browser can render animated GIFs as animations.
ImgJpgBaselineSupported	Boolean	true | false	Set to "true" if the browser supports baseline JPGs.
ImgJpgProgressiveSupported	Boolean	true | false	Set to "true" if the browser supports progressive JPGs.
ImgLocalsrcSupported	Boolean	true | false	Set to "true" if the device has a locally stored image library and can access these images using the wml localsrc attribute of the img tag.
ImgPNGSupported	Boolean	true | False	Set to "true" if the browser supports PNG format images.
ImgSVGSupported	Boolean	true | False	Set to "true" if the browser supports SVG format images.
ImgTypePref	String	.gif .wbmp	A comma delimited list (no spaces) of preferred image types for the browser, for example .gif, .wbmp.
ImgWBMPSupported	Boolean	true | false	Set to "true" if the browser supports WBMP format images.

ImgZeroBorderDefeatsNavigation	Integer	"true" =Navigation border is invisible on this device if img link border is set to "0" "false" =Navigation border is not affected by img link border setting	On certain devices, setting border="0" results in the image link navigation border being invisible. For these devices, border must be set to "1".
IsDeviceRoot	Boolean	true | false	Set to "true" if the device profile represents the initial version of a real-world device and not an emulator class of devices.
IsFullBrowser	Boolean	true | false	Set to "true" if large browser.
IsLandscapePDA	Boolean	true | false	Set to "true" if a page designed with a landscape orientation is more suitable for the device.
IsMenuDriven	Boolean	true | false	Set to "true" if a menu-driven design is most suitable for the device.
IsPDA	Boolean	true | false	Set to "true" if the device is a PDA browser.
IsPortraitPDA	Boolean	true | false	Set to "true" if a page designed with a portrait orientation is more suitable for the device.
Is3GCapable	Boolean	true | false	Set to "true" if the device supports 3G connectivity.
J2MEDownloadLimit	Integer	64000	Max size in bytes of the J2ME JAR that can be downloaded by the device.
J2MESupported	Boolean	true | false	Set to "true" if the device supports J2ME.
MLVersion	String	WML1.3 xHTML MP	Comma delimited list (no spaces) that specifies the markup languages the device supports.
MMSReceiveSupported	Boolean	true | false	Set to "true" if the device can receive MMS messages.
MMSSendSupported	Boolean	true | false	Set to "true" if the device can

			send MMS messages.
MMSSupported	Boolean	true | false	Set to "true" if the device is MMS capable.
MP3Supported	Boolean	true | false	Set to "true" if the device can handle MP3 format.
MaxImageHeightPixels	Integer	21	Maximum height in pixels.
MaxImageSize	Integer	2600	Maximum size of an image in bytes that can be received.
MaxImageWidthPixels	Integer	50	Maximum image width in pixels.
MaxObjectsInMessage	Integer	3	Maximum objects in a message.
MaxTextSize	Integer	102400	Maximum Text Size.
MaxWapDeckSize	Integer	2800	Maximum deck size, in bytes, that a device can receive.
MexeSupported	Boolean	true | false	Set to "true" if the device supports MexE.
MultipartPreferred	Boolean	true | false	Set to "true" if the device prefers multipart content.
NetworksSupported	String	GSM1900 GSM1800 GPRS	Comma delimited list (no spaces) of network technologies supported by the device.
PDFSupported	Boolean	true | false	Set to "true" if the device supports PDFs.
PreferTablesForNavList	Boolean	true | false	Set to "true" if the device is able to properly support the tables created in navigational menu styling.
PreferredCharsets	Sting	UTF-8;Q=0.8,ISO-8859-1	Indicates the preferred character sets for the device.
ProtectWrappingContentTypeList	String	application/ vnd.oma.drm.message	Indicates the content types the device supports Protect Wrapping for.
RecognitionCheckMeBefore	String	SiemansS55, MotorolaV60	Specifies a list of devices that a given device should take precedence over in the device recognition process.

RecognitionHeaders	String / regex (regex applies to the pattern only)	Accept-charset:utf-8 or Accept-charset:.*iso.*	Specifies a list of headers (additional to the User-Agent header) that need to be checked during device recognition. This list will also contain the pattern, which may be a substring or regular expression, to search those headers for.
RecognitionUAPattern	String / regex	Nokia3650 or ^Nokia3650.*	Specifies a list of regular expressions or strings to match against the User-Agent header. NOTE: If it contains a regex, then the "RecognitionRequiresRegex" attribute (below) must also be set to true.
RecognitionRequiresRegex	Boolean	true | false	Set to "true" if the RecognitionUAPattern and RecognitionHeaders (patterns) attributes should be interpreted as a regex pattern (Regular Expression).
RingtoneDownloadSupported	Boolean	true | false	Set to "true" if the device can download ringtones.
RingtoneFormatSupported	String	midi, i-Melody	Indicates the ringtone formats supported by the device.
RingtoneMonophonicSupported	Boolean	true | false	Set to "true" if the device can download monophonic ringtones.
RingtonePolyphonicSupported	Boolean	true | false	Set to "true" if the device can download polyphonic ringtones.
RingtonePref	String	rng,midi, amr	An ordered list of preferred ringtone formats.
SMSLongMessagesSupported	Boolean	true | false	Set to "true" if the devicecan support SMS messages longer than 160 characters.
ScreenOrientation	String	Portrait Landscape	Specifies whether the device has a portrait (most devices) or landscape (communicators) orientation.

ScreenSaverSupported	Boolean	true | false	Set to "true" if the device can support screensavers.
SmartMessagingSupported	Boolean	true | false	Set to "true" if the device supports Smart Messaging.
StreamingAudioCodecsSupported	String	AMR,AWB,AAC	Comma delimited list (no spaces) of streaming audio codecs supported by the device.
StreamingVideoCodecsSupported	String	MPG4,WMV,H263,RV	Comma delimited list (no spaces) of streaming video codecs supported by device.
SupportsAbsoluteWidth	Boolean	true | false	Set to "true" if the device supports absolute widths on images.
SupportsCSS	Boolean	true | false	Set to "true" if the device supports Cascading style Sheets.
SupportsRelativeWidth	Boolean	true | false	Set to "true" if the device supports relative widths on images.
SyncMLSupported	Boolean	true | false	Set to "true" if the device has support for SyncML.
TableRowsFunctionAslink	Boolean	true | false	Set to "true" if the device browser renders table rows as links automatically.
TextBrowser	Boolean	true | false	Set to "true" if the device browser can only render text and not images.
TextColumns	Integer	15	Maximum number of text columns that the screen can accommodate.
TextFormatBigSupported	Boolean	true | false	Set to "true" if plain text wrapped in <big> tags is rendered in big font.
TextFormatBoldSupported	Boolean	true | false	Set to "true" if plain text wrapped in bold <b> tags is rendered in bold font.
TextFormatEmphasisSupported	Boolean	true | false	Set to "true" if plain text wrapped in <emphasis> tags is entered in an emphasized font.

TextFormatItalicSupported	Boolean	true | false	Set to "true" if plain text wrapped in italics <i> tags is rendered in italic font.
TextFormatSmallSupported	Boolean	true | false	Set to "true" if plain text wrapped in <small> tags is rendered in small font.
TextFormatStrongSupported	Boolean	true | false	Set to "true" if plain text wrapped in <strong> tags is rendered in a strong font.
TextFormatUnderlineSupported	Boolean	true | false	Set to "true" if plain text wrapped in underline tags is rendered with an underline.
TextRows	Integer	3	Number of rows that the device-screen can accommodate using the device system font.
TitleRow	Boolean	true | false	Specifies whether the device has a title row.
TouchScreenSupported	Boolean	true | false	Set to "true" if the device supports touch-screen input.
URLRequestLength	Integer	256	Max length of URL request.
USSDSupported	Boolean	true | false	Set to "true" if the device supports USSD technology.
UsableHeightPixels	Integer	570	Screen height excluding items like scroll bars.
UsableWidthPixels	Integer	770	Screen width excluding items like scroll bars.
UseTablesForNavList	Boolean	true | false	Set to "true" if tables should be used for navigation list styling.
UseUAProf	Boolean	true | false	Set to "true" if a manufacturer UAProf file is available for the device.
VideoSupported	String	mpeg	Comma delimited list (no spaces) of the video formats that the device supports.
VideoTypePref	String	mpeg,mp4	Ordered list of preferred video formats.

ViewableHeight	Integer	30	Screen height in pixels.
ViewableWidth	Integer	80	Screen width in pixels.
WAPPushSISupported	Boolean	true | false	Set to "true" if the device supports WAP Push Service Indication.
WAPPushSLSupported	Boolean	true | false	Set to "true" if the device supports WAP Push service loading.
WAPPushSupported	Boolean	true | false	Set to "true" if the device supports WAP Push.
WAPVersion	String	1.2.1	Specifies version of WAP supported by the device.
WMLScriptSupported	Boolean	true | false	Set to "true" if the device supports WML Script.
WMLVersion	String	1.3	Specifies which version of WML the device supports.
WTAIInternationalPrefix	String	+00	Indicates the international prefix that should be used when specifying telephone numbers.
WTAIMakePhoneCallSupported	Boolean	true | false	Set to "true" if the device has phone dialing capabilities.
WTLSSupported	String	WTLS_Class1	Indicates the WTLS class supported by the device.
WavEncodingsSupported	String	PCM8	Indicates the supported Wav file encodings.

Deprecated Device Attributes

This is a list of the deprecated items in the Device Repository. These attributes are still functional for the purpose of backward compatibility although it is recommended that you use the alternative if available.

The attribute that should be used as a replacement is listed each deprecated attribute name. Each of these new attributes should be prefixed with "UAProf." to form the complete name.

Deprecated device attributes

Deprecated attribute name	Data type	Example values	Description
Acceptheader	String	text vnd.wap.wml image vnd.wap.wbmp	Comma delimited list (no spaces) used to specify the media types that are acceptable for the response (that is, what can be sent to the browsing device). Replaced by: SoftwarePlatform.CcppAccept
AudioFormatSupported	String	mid au wav mp3	Comma delimited list (no spaces) of audio formats the device is capable of supporting. Replaced by: SoftwarePlatform.CcppAccept
CDC1xSupported	Boolean	true | false	J2ME Connected Device Configuration. Replaced by: SoftwarePlatform.JavaPlatform
CLDC1xSupported	Boolean	true | false	J2ME Limited Device Configuration. Replaced by: SoftwarePlatform.JavaPlatform
CharsetSupported	String	utf8 ascii ISO8859-1	Comma delimited list (no spaces) of character sets supported. Replaced by: SoftwarePlatform.CcppAccept-Charset
ColorDepth	Int	12	Indicates the number of bits per pixel supported. Replaced by: HardwarePlatform.BitsPerPixel
ColorType	String	Colour	Specifies whether the screen is black & white, color or grayscale. Replaced by: HardwarePlatform.ColorCapable
DeviceClass	String	PDA FULLBROWSER WMLBROWSER	Describes the category of device. Replaced by: IsPDA, IsPortraitPDA, IsLandscapePDA IsFullBrowser IsMenuDriven

Appendixes

EmailClient	String	POP3 SMTP	Comma delimited list (no spaces) that indicates the e-mail protocols that the device supports. Replaced by: SoftwarePlatform.Email-URI-Schemes
FoundationProfile1xSupported	Boolean	true | false	Java (CDC) profile. Replaced by SoftwarePlatform.JavaPlatform
ImageFormatSupported	String	wbmp bmp gif animgif png jpeg	Comma delimited list (no spaces) of all of the image formats supported by the device, for example, gif, wbmp and png. Replaced by: SoftwarePlatform.CcppAccept
ImgMapTransformEnabled	Boolean	true | false	Set to "true" if image maps are to be transformed into links. No replacement.
ImgMapTransformShowImage	Boolean	true | false	Set to "true" if images are also delivered with an image map. No replacement.
JavaPhone1xSupported	Boolean	true | false	Used by some devices with Personal Java. Replaced by: SoftwarePlatform.JavaPlatform
JavaScriptSupported	Boolean	true | false	Set to "true" if the device supports JavaScript. Replaced by: BrowserUA.JavaScriptEnabled
MIDP1xSupported	Boolean	true | false	Set to "true" if the device supports J2ME (CLDC) MIDP Profile Version 1. Replaced by: SoftwarePlatform.JavaPlatform
MIDP2xSupported	Boolean	true | false	Set to "true" if the device supports J2ME (CLDC) MIDP Profile Version 2. Replaced by: SoftwarePlatform.JavaPlatform
MXImageMapShowImage	Boolean	true | false	Allows you to display links in an image map on a PDA. No replacement.

MXImageTypePref	String	.gif .wbmp	A comma delimited list (no spaces) of preferred image types for the browser. No replacement.
MXListBoxHeight	Int	Any Integer	Default is 6. No replacement.
MultipartSupported	Boolean	true | false	Set to "true" if the device supports accepts multipart content. Replaced by: SoftwarePlatform.CcppAccept
OSVersion	String	4.22, 5.0, and so on.	Indicates the version of the Operation System on the device, where applicable. Replaced by: SoftwarePlatform.OSVersion
OSType	String	AMX, PALM, and so on.	Indicates the Operating System on the device, where applicable. Replaced by: SoftwarePlatform.OSName
PersonalJava1xSupported	Boolean	true | false	Personal Java Specification. Replaced by: SoftwarePlatform.JavaPlatform
ScreenAspectRatioPixels	String	1X1, 1X2, and so on.	Pixels on most devices are higher than wide which explains why sometimes images can look distorted on browsers. The pixel aspect ratio specifies the width to height pixel ratio on a devices display. Replaced by: HardwarePlatform.PixelAspectRatio
SoundHandling	Boolean	true | false	Set to "true" if the device has audio capability. Replaced by: SoftwarePlatform.CcppAccept
TableSupported	Boolean	true | false	Set to "true" if the device supports tables. Replaced by: BrowserUA.TablesCapable
WTAIAddPhoneBookEntrySupported	Boolean	true | false	This is part of WTAI support and allows a selected number to be saved to the devices phone book. Replaced by: WapCharacteristics.WtaiLibraries

Appendix C—Use the Admin Console Tool to Manage Devices and Device Attributes in the Device Repository

When the Device Repository is represented as a database, you will use the Administration Console tool to add, remove and modify devices and device attributes. The Administration Console is a Java-based GUI that provides a convenient way of setting up, retrieving and modifying the attributes associated with each profile.

Quick Start

The following table introduces the basic steps in using the Administration Console.

Administration Console Quick Start Guide

To…	Choose
Launch the console	In BEA WebLogic Workshop, from the Tools Launcher Icon or launch directly from **<BEA_install_directory>/weblogic81/mobility/applications/***AdminConsole.exe* or *AdminConsole*
Login	Apps → Login
Logout of the console	Apps → Logout
Close all windows	Apps → Close All
Refresh the Device Repository	Apps → Refresh Database
Exit the console	Apps → Exit

Log In

The "Administration Console Login" window opens when you launch the application.

Enter the correct WebLogic Mobility Server IP address and web application address in the **Server** field, for example **localhost:8080/<application>/**.

Note: The **Server** field recalls the last four servers that the Administrator successfully connected to.

If required, select the **Password Protected** check box to enable the **Username** and **Password** fields.

If required, enter your username and password in the respective fields. As you type your password the characters appear as asterisks.

Click **Login** to display the "Administration Console" window.

Use the System Monitor

The System Monitor displays the Free Memory available and refreshes the console.

- Choose **Apps → System Monitor**.

Refreshing the Console Automatically

- Set the Refresh interval (in seconds) for the Administration Console using the refresh period indicator

Refreshing the Console Manually

- Click **Refresh** to refresh the Administration Console display

Use the Administration Console Toolbar

The Administration toolbar provides a convenient method for accessing the administrative functions within WebLogic Mobility Server. The following illustrates the tasks associated with each image on the toolbar.

Administration Console Toolbar
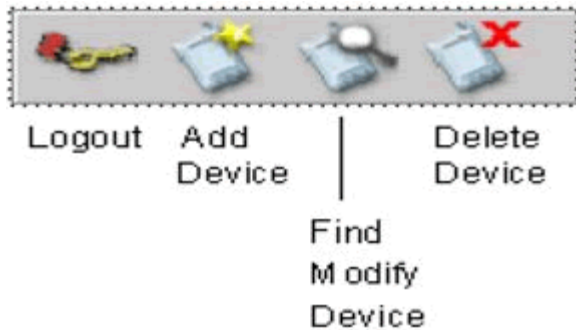
Create and Modify Device Profiles

Device profiles are configured from within the Administration Console. The existing profiles and attributes can be modified, or new ones can be created. This can be useful for capturing more device-specific information to finely tune your content delivery for a specific purpose.

Create Device Profiles

Three steps are required when adding a new device profile to WebLogic Mobility Server:

- Complete the "Basic Details" for the device

- Configure the standard attribute values for the device

- Create new attributes if required

Add a Device

To add a device:

1. Choose **Device** → **Add Device**. Select the parent device class to which this device will belong

2. Complete the details on the **Basic Device Details** tab:

Basic Device Details

Field	Description
Device Name*	Type in a unique name to identify this device or device class
Display Name*	Type in the label you want displayed for this device
Description	Optionally, type in a description of this device

    Note: * Indicates a required field

3. Click **Next** to proceed to the next tab.

4. When adding a device to the database, there is a standard set of attributes that need to be configured for the new device.

Adding a Device Attribute

To add a new Device Attribute:

1. Click **Add** on the **Attributes** tab.

2. Select the **Device** attribute option.

3. In the **Name** field, enter a name for the new attribute.

4. From the "Type List", select a data type for the new attribute. If you've chosen the String data type, and want to restrict its values to a predefined list, enter a comma-separated list of values in the **Permitted Values** field.

Note: The **Modifiable By** option should be ignored. This is a legacy option and has been deprecated.

Configuring an Attribute

1. Select the device that you wish to configure.

2. Click **Next** until you reach the **Attribute Values** tab.

3. Select the attribute you want to configure and double-click in the corresponding **Value** field.

Modifying a Device Profile

1. You can add and delete attributes or change attribute values. Inherited attributes cannot be deleted: the **Delete** button will be unavailable if you select an inherited attribute.

2. Choose **Device** → **Find** and "Modify Device". When the "Device" panel appears, select the device you want to modify. Click **Next** to move between tabs.

3. Click **Finish** when you are satisfied with your changes.

Viewing an Attribute

1. Select the attribute from the Attributes list and then click **View**.

Deleting Devices

You can only delete devices that you have added to the device hierarchy; you cannot delete pre-installed devices.

1. Choose **Device** → **Delete Device**, select the device that you want to delete and click **Delete**.

Appendix D—Configure Device Repository Manager to Connect to the Update Service via a Web Proxy

You can also configure Device Repository Manager to connect to the Device Repository Online Update Service via a web proxy to download the latest updates. To achieve this, open the Device Repository Manager and select **File → Proxy Settings**. In the Proxy Details dialog box that is displayed, set the appropriate proxy settings as illustrated:
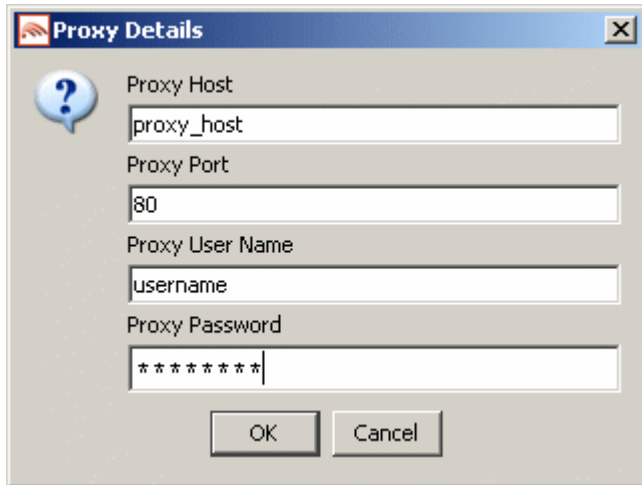
Appendix E—Fallback Recognition Logic Expression Language Details

To allow more than one header to be considered during recognition at any given node we use the **FallbackRecognitionLogic** attribute and its associated expression language. More information on the expression language is provided below.

FallbackRecognitionLogic Associated Expression Language

Expressions are made up of terms and operators. Terms are of the form *HeaderName***:"***Substring***"**, including the quotes. A term evaluates to true if the named header contains the specified substring.

If the substring needs to contain a double-quotes character, it is escaped with a backslash. For a literal backslash, two backslashes are used. For example, to check if the **MyHeader** header contains the string:

> **substring containing " and \ characters**

…you would use …

```
MyHeader:"substring containing \" and \\ characters"
```

To combine terms into complex expressions, FallbackRecognitionLogic supports the logical operators *and*, *or*, and *not*, represented by "**&**" (ampersand), "**|**" (vertical line) and "**!**" (exclamation mark) respectively. Parentheses (round brackets) are supported for grouping terms and specifying precedence. For example:

```
(Accept:"wml" | x-wap-profile-diff:"") & !(User-Agent:"mozilla" | User-
Agent:"Mozilla")
```

Note that in this example the empty substring **""** is used. This term evaluates to true if the named header exists, no matter what its value, and evaluates to false if the header is absent. Therefore the expression above will evaluate to true if the **Accept** header contains the string **wml** or if any **x-wap-profile-diff** header is present, but in either case only if the **User-Agent** header does not contain the strings **mozilla** or **Mozilla**.

Header names are case-insensitive while substring values are case-sensitive.