Getting Started with phoneME Feature Software

This guide contains the following chapters:

Introduction

This guide contains information for building and running the phoneME™ Feature software, an optimized Java ME stack based on Milestone Release 1 (MR1) of the phoneME Feature project.

This guide allows you to take the following general steps:

- Set up and configure your Linux/i386 or Win32/i386 build environment.
- Build a default version of the PCSL libraries, CLDC, and phoneME Feature software binaries.
- Run the phoneME Feature software on your default platform.
- Find additional information about phoneME Feature software.

Note: MR1 is the stable version of phoneME Feature software. It is unlikely to change beyond adding necessary bug fixes. For active development, see the MR2 version of the phoneME Feature project.

What's in This Release

The phoneME Feature software contains the following software features and components:

- Support for Texas Instruments P2SAMPLE64-V6 reference platform (P2 board)
- Multitasking capabilities and support for multiple, simultaneous applications (MIDlets)

- Full or partial implementations for the following Java Specification Request (JSR) optional packages:
    - Connected Limited Device Configuration (CLDC) 1.0 (JSR 30) and 1.1. (JSR 139)
    - Mobile Information Device Profile (MIDP) 1.0 (JSR 37) and 2.0 (JSR 118)
    - Java Wireless Messaging (WMA) 1.1 (JSR 120) and 2.0 (JSR 205)
    - Personal Information Management (PIM) 1.0 (JSR 75)
    - FileConnection 1.0 (JSR 75)
    - Java APIs for Bluetooth Wireless Technology 1.0 (JSR 82)
    - Mobile Media API (JSR 135)
    - J2ME Web Services 1.0 (JSR 172)
    - Security and Trust Services API (SATSA) 1.0 (JSR 177)
    - Java Technology for the Wireless Industry version 1.0 (JSR 185)
    - Scalable 2D Vector Graphics API for J2ME 1.1 (JSR 226)
- Customizable user interface (skin) using Chameleon Adaptive User Interface Technology (AUIT) and XML

What's Not in This Release

The following software components are not included in the phoneME Feature software release:

- Mobile 3D Graphics API (JSR 184)
- Audio functionality in MIDP 2.0 (JSR 118), Mobile Media APIs (JSR 135), and Java Technology for the Wireless Industry (JSR 185) implementations

---

Commercial Documentation

This guide provides only the minimal information needed to set up, build, and run the phoneME Feature software quickly, using only the default settings for each implementation.

For more information about phoneME Feature software topics, including additional build options, porting, multitasking, using Adaptive User Interface technology, and others, see the commercial documentation set.

Build Environment

This document contains the following sections:

The phoneME Feature software build environment must be set up on your *build platform*. The build platform is the machine where source code is installed, the compilers are run, and executable code is built. This executable code is then installed and run on a *target platform*.

The phoneME Feature software supports the following build platforms:

- Linux/i386
- Win32/i386

Note: Linux/i386 is not supported as a target platform.

The phoneME Feature software supports the following target platforms:

- Linux/ARM (Texas Instruments P2SAMPLE64-V6 board)
- Windows/i386 (which runs the phoneME Feature software in emulation mode)

Requirements for Building on a Linux Platform

To properly build executables for the Linux/ARM target platform, a Linux/i386 build platform must meet the following requirements:

- Red Hat Linux distribution version 7.2 - 9.0
- Java 2 Platform, Standard Edition (J2SE™) Development Kit (JDK™) version 1.4.2
- GNU Make version 3.79.1 or later
- GNU Cross Compiler (GCC) 3.4.6 or later
- Doxygen version 1.4.1
- Development Kit for the Java Card™ Platform 2.2.1

Requirements for Building on a Windows Platform

A Win32/i386 build-target platform must meet the following requirements:

- Windows 2000 or Windows XP
- JDK version 1.4.2
- GNU Make version 3.79.1 or later
- Microsoft Visual C++ (MSVC++) Professional Version 6.0 or higher (or Visual Studio .NET 2003 or later)
- Doxygen version 1.4.4
- Microsoft Processor Pack 5, version 6.15 or later
- Cyg4Me version 1.1 (a version of Cygwin for building Java ME technology. Cyg4Me contains GNU Make.)
- Development Kit for the Java Card Platform 2.2.1

`make` Targets and Options

The phoneME Feature software and its supporting components (PCSL and CLDC) are built using the makefiles and source files provided for each component, using only the component's default settings.

To build a component, set the environment variables and enter the `make` command:

```
$ make all
```

Preparing Your Build Environment

Preparing your environment requires you to set environment variables for your Linux and Windows build platforms. Other environment variables may be needed, specific to the component being built. These are described in the chapter for each component.

BuIld environment variables can be set in two ways:

- As a system variable

- On the command line, when you run `make`

Note: This document assumes the use of the Bash shell on the Linux platform. If you are using some other shell, set the variables for the shell you are using.

Setting Variables for a Linux Platform

To set a build environment variable as a system variable for a Linux platform, use the following format:

```
$ export VARIABLE=value
```

For example, to set `MIDP_DIR` as a system variable, type:

```
$ export MIDP_DIR=InstallDir/jwc/midp
```

Note: *InstallDir* is the location of your phoneME Feature software.

To set a build environment variable using the `make` command line for a Linux platform, use the following format:

```
$ make VARIABLE=value make_target
```

For example, to set `MIDP_DIR` on a `make` command line, type:

```
$ make MIDP_DIR=InstallDir/jwc/midp all
```

Setting Variables for a Windows Platform

Note: This document assumes the use of Cyg4Me on the Windows platform.

To set a build environment variable as a system variable for a Windows platform, use the following format:

```
$ set VARIABLE=value
```

For example, to set the `MIDP_DIR` as a system variable, type:

```
$ set MIDP_DIR=InstallDir/jwc/midp
```

To set a build environment variable on the `make` command line for a Windows platform, use the

following format:

```
$ make VARIABLE=value make_target
```

For example, to set `MIDP_DIR` on a `make` command line, type:

```
$ make MIDP_DIR=InstallDir/jwc/midp all
```

Setting Up the Linux Build Environment

To set up the Linux/i386 build environment, you must do the following things:

- Acquire Monta Vista Developer Tools
- Set Linux platform environment variables

Acquiring Monta Vista Developer Tools

To build phoneME Feature software for the Linux/ARM (P2 board) target platform, you must acquire the MontaVista CEE 3.1 ADK developer tools.

Instructions for use of these tools are provided by MontaVista.

Setting Environment Variables

To properly set up the Linux/i386 build environment, you must set the environment variables shown in TABLE 2-1.

TABLE 2-1 Linux Platform Environment Variables

Name	Description
`JDK_DIR`	Directory that contains the JDK release. For example, `/usr/java/j2sdk1.4.2`.
`MIDP_DIR`	Top-level MIDP directory, usually *InstallDir*`/jwc/midp`.

`MIDP_OUTPUT_DIR`	Location where output from phoneME Feature build is stored.
`LD_LIBRARY_PATH`	Directory that contains the libraries needed by the build and run the phoneME Feature software.
`GNU_TOOLS_DIR`	Build tools for the Linux/i386 platform, used to generate executables for the Linux/ARM target platform.

Setting Up the Windows Build Environment

To set up your Windows platform, you must take the following steps:

- Install Cyg4Me
- Set environment variable
- Update your PATH environment variable

Installing Cyg4Me

Before you build phoneME Feature on a Windows platform, download and install Cyg4Me, version 1.1. Cyg4Me is a standardized version of Cygwin for building Java ME platform products on Windows. It provides the necessary UNIX® system tools, such as `sh` and `gnumake`.

1. Download the Cyg4Me zip file `cyg4me1_1_full.zip`.

   This file contains all the executables and libraries needed to build on the Windows platform.

2. Unpack the file into a directory.

   For example, you could unpack the file into `C:\cyg4me`.

3. Ensure that the Cyg4Me `bin` directory is the only version of Cygwin on your path.

   See Updating Your PATH Environment Variable for instructions.

Setting Environment Variables

To build the phoneME Feature software on a Windows platform, you must set the environment variables shown in TABLE 2-2.

TABLE 2-2 Windows Platform Environment Variables

Name	Description
`JDK_DIR`	Directory that contains the JDK release. For example, C:`/java/j2sdk1.4.2`.
`MIDP_DIR`	Top-level MIDP directory, usually *InstallDir*`/jwc/midp`.
`MIDP_OUTPUT_DIR`	Location where output from phoneME Feature build is stored.

Updating Your PATH Environment Variable

If you are using a Windows system, you must put Cyg4Me on your `PATH` environment variable.

1. Make the `bin` directory of your Cyg4Me installation the first element of your `PATH` variable's value.

   For example, you could set your `PATH` in the command window where you build the Java ME platform products with this command:

   ```
   $ set PATH=C:\cyg4me\bin;%PATH%
   ```

2. Remove all `PATH` elements that point to other versions of Cygwin.

   This is necessary because Cyg4Me contains binaries from a specific snapshot of Cygwin. It might not be compatible with other versions of Cygwin installed on the same PC.

   If you performed Step 1 in a command window, perform this step in the same window.

Building a PCSL Reference Port

This document contains instructions for building a PCSL reference port. It has the following sections:

The phoneME Feature software uses the PCSL libraries to build the phoneME Feature software. PCSL contains the following individual services:

- file
- memory
- network
- print

Note: Although each PCSL service can be individually built, this guide describes how to build the full PCSL only.

To build a default implementation of PCSL, you take the following general steps:

- Set environment variables for your platform
- Run `make all` to build the PCSL implementation
- Build Doxygen PCSL documentation (optional)

For more information on building individual PCSL services and other build options, see the *Sun Java Wireless Client Build Guide*.

PCSL Environment Variables

Note: Make a note of the values you choose for these variables. When building the phoneME Feature software, you must set corresponding values.

PCSL has three environment variables that can be set, as shown below:

- `PCSL_PLATFORM` - Identifies the target operating system, the target CPU, and the

compiler that the build system uses to create the PCSL library. Its value has the form *os_cpu_compiler*. This is a mandatory variable and must be set when building for all platforms.

- `PCSL_OUTPUT_DIR` - Specifies into which directory the build system puts its output (for example, object files and libraries). This is an optional variable. If it is not specified, PCSL sets the output directory for you.

- `GNU_TOOLS_DIR` - Specifies the location of the Monta Vista build tools for the Linux platform.

Note: In order for the CLDC and phoneME Feature software build processes to find PCSL output, the variable `PCSL_OUTPUT_DIR` *must* be set as a system variable.

Setting System Variables on a Linux Platform

Note: This document assumes the use of the Bash shell on the Linux platform. If you are using some other shell, set the variables for the shell you are using.

To set `PCSL_PLATFORM`, `PCSL_OUTPUT_DIR`, and `GNU_TOOLS_DIR` as Linux system variables, enter the following commands:

```
$ export PCSL_PLATFORM=linux_arm_gcc
$ export PCSL_OUTPUT_DIR=$HOME/pcsl_output
$ export GNU_TOOLS_DIR=/opt/montavista/cee/devkit/arm/v4t_le/
armv4tl-hardhat-linux
```

Note: In the variables shown above, `$HOME` represents the location of your PCSL software. *pcsl_output* represents the location of the PCSL build output for your target platform (for example, `linux_arm`).

Setting System Variables on a Windows Platform

Note: This document assumes the use of Cyg4Me on the Windows platform.

To set `PCSL_PLATFORM` and `PCSL_OUTPUT_DIR` as Windows system variables, enter the following commands:

```
$ set PCSL_PLATFORM=win32_i386_vc
$ set PCSL_OUTPUT_DIR=C:/my_output/pcsl/output
```

Note: In the variables shown above, *output* represents the location of the PCSL build output for your target platform (for example, `win32_i386)`.

Using PCSL `make` Files

To build a full implementation of PCSL, you run the `make` command at the top of your PCSL file tree. The `make` command calls a platform-specific makefile, called *os_cpu_compiler*`.gmk`, which builds an implementation of PCSL for your specific platform.

Each target platform has a makefile, which is tailored to the target platform's operating system, CPU, and the compiler used to build the PCSL implementation. For example, the makefile for the Windows implementation of PCSL is called `win32_i386_vc.gmk`.

The file *os_cpu_compiler*`.gmk` is located in the directory *InstallDir*`/pcsl/makefiles/platforms`.

Note: *InstallDir* is the location of your phoneME Feature software.

PCSL Build Targets

The PCSL build system provides several `make` targets that enable you to build specific sections of the PCSL reference port. However, this guide describes only two, as shown in TABLE 3-1.

TABLE 3-1PCSL Build Targets

Name	Description
`all`	Bulds libraries and public header files for services. Build results are in the `$PCSL_OUTPUT_DIR/`*`os_cpu`* subdirectory.
`doc`	Builds services API document generated by Doxygen. Output is in the `$PCSL_OUTPUT_DIR/doc` subdirectory. For more information, see Building PCSL Documentation.

For more information on PCSL build targets, see the *Sun Java Wireless Client Build Guide*.

Building PCSL Software

PCSL software is used to build both CLDC and phoneME Feature software. To build CLDC for the Linux/ARM target platform, you must first build PCSL for the Linux/i386 platform. By following the procedures in this section, you can build a default PCSL software reference port that contains libraries, tools, tests, and documentation bundles for your target platform.

Note: The following procedures build more than just the *lib* and *inc* directories. However, these are all the only directories needed for software development. For a complete list of directories built on the Linux and Windows platforms, see the *Sun Java Wireless Client Build Guide*.

Building for a Linux/i386 Platform

1. Change to the PCSL directory.

   ```
   $ cd InstallDir/pcsl
   ```

2. Run the `make` command.

   ```
   $ make GNU_TOOLS_DIR= PCSL_PLATFORM=linux_i386_gcc all
   ```

   Note: In the command above, the `GNU_TOOLS_DIR` variable is intentionally left undefined. This lack of definition overrides the previously set variable and prevents the build from picking up the Monta Vista tools used by other Linux platform builds.

   When the `make` command has successfully completed, the following directories exist:

   - `$PCSL_OUTPUT_DIR/linux_i386/lib`
   - `$PCSL_OUTPUT_DIR/linux_i386/inc`

Building for a Linux/ARM Platform

1. Change to the PCSL directory.

   ```
   $ cd InstallDir/pcsl
   ```

2. Run the `make` command.

   ```
   $ make all
   ```

When the `make` command successfully completes, the following directories exist:

- `$PCSL_OUTPUT_DIR/linux_arm/lib`
- `$PCSL_OUTPUT_DIR/linux_arm/inc`

Building on a Windows Platform

1. Mount the phoneME Feature software's PCSL directory at the root level.

   For example, if you installed the phoneME Feature software in `C:\jwc1.1.3`, use this command:

   ```
   $ mount C:\jwc1.1.3\pcsl /jwc1.1.3/pcsl
   ```

2. Change to the PCSL directory.

   `$ cd` *InstallDir*`/pcsl`

3. Run the `make` command.

   `$` make all

   When the `make` command has successfully completed, the following directories exist:

   - `$PCSL_OUTPUT_DIR/win32_i386/lib`
   - `$PCSL_OUTPUT_DIR/win32_i386/inc`

Building PCSL Documentation

If you installed the Doxygen tool (from http://www.doxygen.org), you can create HTML documents from porting interfaces. To do this for PCSL, use the `make doc` build target. The documents are put in the directory `$PCSL_OUTPUT_DIR/doc/doxygen/html`.

Note: The build system assumes that the Doxygen executable is `/usr/bin/doxygen`. If your executable is in a different location, edit the `Docs.gmk` file in the directory *InstallDir*`/pcsl/makefiles/share` (where *InstallDir* is the location of your phoneME Feature software).

Set the `DOXYGEN_CMD` variable to the appropriate value.

Generating Doxygen Documentation

The following procedure for building Doxygen documentation is the same for both Linux and Windows build platforms.

1. Change to the PCSL directory:

   ```
   $ cd InstallDir/pcsl
   ```

2. Run the `make doc` command:

   ```
   $ make doc
   ```

Viewing PCSL Documents

Use any browser to display the Doxygen output file at the following URL:

```
file:///$PCSL_OUTPUT_DIR/doc/doxygen/html/index.html
```

Building a CLDC Reference Port

CLDC software is an OSS community version of Sun Microsystems' Connected Limited Device Configuration HotSpot™ Implementation virtual machine. Although primarily created from the same shared code base, Sun Microsystems' commercial product might not fully and accurately represent the OSS source base, due to licensing and other legal restrictions.

This document has the following sections:

For more information about CLDC, see CLDC HotSpot Implementation software documentation.

CLDC Build Environment

This chapter provides basic instructions for building the CLDC software, which is needed to build the phoneME Feature software. Many CLDC software build configurations and options are possible. This guide describes how to build a default implementation only.

Setting Variables for a Linux Platform

Note: This document assumes the use of the Bash shell on the Linux platform. If you are using some other shell, set the variables for the shell you are using.

To properly build CLDC software on a Linux build platform, you must set the environment variables shown in TABLE 4-1. For more information on setting other Linux build environment variables, see Setting Up the Linux Build Environment.

Once the variables in TABLE 4-1 are set, no other Linux build environment set up is needed.

TABLE 4-1 Linux Platform Environment Variables

Name	Description

`JVMWorkSpace`	The location of your CLDC software source code workspace.
`JVMBuildSpace`	The location of your CLDC software build output.
`CLDC_DIST_DIR`	Required for building phoneME Feature software. Must point to the location of CLDC software build output. For example: `CLDC_DIST_DIR=$JVMBuildSpace/linux_arm/dist`
`PATH`	Must be set to include all gnumake and compiler tools used in the build process.
`ENABLE_PCSL`	Required for building the phoneME Feature software. Must be set to `true`.
`ENABLE_ISOLATES`	Required for using multitasking functionality. Must be set to `true`.

For example, to set the variable `JVMWorkSpace` as a system variable, enter this command:

```
$ export JVMWorksSpace=/development/openvm/linux
```

To set `JVMWorkSpace` on a `make` command line, enter this command:

```
$ make JVMWorkSpace=/development/openvm/linux
```

Setting Variables for a Windows Platform

Note: This document assumes the use of Cyg4Me on the Windows platform.

To properly build the CLDC software on a Windows platform, you must set the environment variables shown in TABLE 4-2. For more information on setting other Windows build environment variables, see Setting Up the Windows Build Environment.

Once the variables shown in TABLE 4-2 are set, no other build environment setup is needed.

TABLE 4-2 Windows Platform Environment Variables

Name	Description
JVMWorkSpace	The location of your CLDC software source code workspace.
JVMBuildSpace	The location of your CLDC software build output.
CLDC_DIST_DIR	Required for building phoneME Feature software. Must point to the location of CLDC software build output. For example: CLDC_DIST_DIR=$JVMBuildSpace/win32_i386/dist
PATH	Must be set to include all gnumake and compiler tools used in the build process.
INCLUDE	Points to the include directory inside your MSVC++ installation.
LIB	Points to the lib directory inside your MSVC++ installation.
X86_PATH	Set it to the same value as your PATH variable.
X86_INCLUDE	Set it to the same value as your INCLUDE variable.
X86_LIB	Set it to the same value as your LIB variable.
ENABLE_PCSL	Required for building the phoneME Feature software. Must be set to true.
ENABLE_ISOLATES	Required for using multitasking functionality. Must be set to true.

For example, to set JVMWorkSpace as a system variable, enter this command:

```
$ set JVMWorkSpace=C:/development/openvm/win32
```

To set `JVMWorkSpace` on a `gnumake` command line, enter this command:

```
$ make JVMWorkSpace=C:/development/openvm/win32
```

Building CLDC Software

The default CLDC software build mode is *Debug*. Building a default version of the CLDC software creates a debuggable version of the CLDC software executable (`cldc_vm_g` on Linux and `cldc_vm_g.exe` on Windows). The debug build version executes relatively slowly, but is valuable for testing and debugging the system.

After building the CLDC system, follow the instructions in Running CLDC Software to run it.

Building CLDC for a Linux Platform

Note: This document assumes the use of the Bash shell on the Linux platform. If you are using some other shell, set the variables for the shell you are using.

1. Ensure that you have set the required environment variables.

   For more information, see Setting Variables for a Linux Platform.

2. Change directory to `$JVMWorkSpace/build/linux_arm` and run the `make` command.

   ```
   $ make debug
   ```

   Output is generated under `$JVMBuildSpace/linux_arm/target/bin`

Building CLDC for a Windows Platform

Note: This document assumes the use of Cyg4Me on the Windows platform.

1. Ensure that you have set the required environment variables.

   For more information, see Setting Variables for a Windows Platform.

2. Change directory to `%JVMWorkSpace%/build/win32_i386` and run the `make`

command.

```
$ make debug
```

Output is generated under `%JVMBuildSpace%/build/win32_i386/target/bin`.

Building CLDC Documentation

You can create HTML documents from CLDC porting interfaces. To do this for the CLDC software, use the `make docs_html` build target.

Generating Javadoc Tool Documentation

Building Javadoc™ tool documentation is the same for both Linux and Windows platforms.

1. Change to the default build directory for your target platform.

   ```
   $ cd $JVMWorkSpace/build/platform
   ```

2. Build Javadoc tool HTML documentation.

   ```
   $ make docs_html
   ```

   The generated HTML documents are put in the following directory:

   ```
   $JVMBuildSpace/doc/javadoc/html
   ```

Viewing phoneME Feature Documents

Use any browser to display the Javadoc tool output file at the following URL:

```
file:///$JVMBuildSpace/doc/javadoc/html/index.html
```

Running CLDC Software

Once you build a debug version of CLDC, you can invoke it from the command line to run a class

compiled from the Java programming language. The path to the executable depends on the target platform (operating system and processor) for which you build the software.

Note: Many more command options are available to run the CLDC software, on both Linux and Windows platforms, than shown here. For more information, see *CLDC HotSpot Implementation Build Guide*.

Running CLDC on a Linux Platform

1. Change to the CLDC build space for the Linux/ARM platform:

   ```
   $ cd $JVMBuildSpace/linux_arm
   ```

2. Enter the following command:

   `$ bin/cldc_vm_g -classpath` *location-of-compiled-applications*`/classes` *classname*

Running CLDC on a Windows Platform

1. Change to the CLDC build space for the Win32/i386 platform:

   ```
   $ cd %JVMBuildSpace%/win32/i386
   ```

2. Enter the following command:

   `$ bin/cldc_vm_g.exe -classpath` *location-of-compiled-applications*`/classes` *classname*

Building a phoneME Feature Software Reference Port

The phoneME Feature software is an Open Source version of Sun Microsystems' commercial implementation, the Sun Java Wireless Client software, version 1.1.3. The phoneME Feature software is built on top of CLDC, an Open Source version of Sun Microsystems' Connected Limited Device Configuration, HotSpot Implementation, version 1.1.3.

This document has instructions for building a default configuration of the phoneME Feature software. It has the following sections:

For proper building and performance, your system must meet the requirements provided in Build Environment

For more information on items in this chapter or other phoneME Feature software build topics, see the *Sun Java Wireless Client Build Guide*.

Using the `make` Command

To build a default implementation of the phoneME Feature software, you run the `make` command with the file, `Default.gmk`. This builds a default version of the phoneME Feature software for your target platform.

The `Default.gmk` file is located in the directory *InstallDir*`/midp/build/`*platform*

Note:*InstallDir* is the location of your phoneME Feature software.

For more information on using the `Default.gmk` file to build a default configuration of the phoneME Feature software for your target platform, see Building phoneME Feature Software.

phoneME Feature Software Build Targets

The phoneME Feature software provides several `make` targets for building a phoneME Feature reference port. Although several `make` targets are available, this guide describes only two, as shown in TABLE 5-1.

TABLE 5-1 phoneME Feature Software Build Targets

Target	Description
`all`	Creates the executables, classes, and tools for a version of the phoneME Feature software.
`docs_html`	Generates phoneME Feature API documentation.

For more information on phoneME Feature build targets, see the *Sun Java Wireless Client Build Guide.*

Default Build Configuration

The default build configuration for your target platform is named using a combination of *operating system* (Linux or Win32), *platform* (ARM or i386), and one or more build *options*. Setting (or not setting) specific build options determines the configuration of the build.

For example, for the Linux/ARM target platform (P2 board), the default build configuration uses Frame Buffer graphics (`fb`), with Adaptive User Interface Technology (`chameleon`), and multitasking (`mvm`). Therefore, the default build configuration is named `linux_arm_fb_chameleon_mvm`.

On the Win32/i386 platform, the default build configuration is `win32_i386`.

For more information about build options in the phoneME Feature software default build configuration, see Default Bulld Configuration Options.

Building a Complete Audio Implementation

In order to build a complete implementation of JSR 135, the following files need to be modified.

On a Win32/i386 platform:

- `jsr135/win32-jsr135/src/native/AMRDecoder.c`
- `jsr135/win32-jsr135/src/native/qsoundnative.c`

On a Linux/ARM platform:

- `jsr135/linux-abb/src/native/qsoundnative.c`

These files contain stubbed implementations of all native methods for the following classes.

On a Win32/i386 platform:

- `com.sun.mmedia.AMRDecoder`

On Win32/i386 and Linux/ARM platforms:

- `com.sun.mmedia.QSound*`

Sun Microsystems' compliant Mobile Media API (JSR 135) implementation utilizes audio libraries licensed from QSound Labs, Inc. (www.qsound.com).

Note: Other libraries are possible. If you choose other audio libraries, you will need to modify the listed files for those libraries.

Building phoneME Feature Software

By following the procedures provided in this section, you can build a default phoneME Feature software reference port that contains the libraries, tools, tests, and documentation bundles for your target platform.

Building for a Linux Platform

Note: This procedure assumes the use of the Bash shell on the Linux platform.

1. Change to the default phoneME Feature software build directory for the Linux/ARM target platform.

```
$ cd InstallDir/midp/build/linux_fb_gcc
```

Note: *InstallDir* represents the location of your phoneME Feature software.

2. Ensure that all environment variables have been set.

   For more information on environment variables, see Build Envrionment and Building a PCSL Reference Port.

3. Ensure that a PCSL reference port has been built for the Linux/ARM target platform.

   For more information on building PCSL, see Building a PCSL Reference Port.

4. Ensure that a CLDC reference port has been built for the Linux/ARM target platform. For more information on building CLDC, see Building a CLDC Reference Port.

5. Use `make` to build a reference port of the phoneME Feature software.

   ```
   $ make VAR_IMPORT_FROM=Default.gmk all
   ```

   This builds the full phoneME Feature software.

   Note: To build Javadoc documentation at the same time you build the phoneME Feature software default build configuration, enter the following command in Step 5:

   ```
   $ make VAR_IMPORT_FROM=Default.gmk docs_html all
   ```

Building for a Windows Platform

Note: This document assumes the use of Cyg4Me on the Windows platform.

1. Change to the default phoneME Feature build directory for the Win32/i386 target platform.

   ```
   $ cd InstallDir/midp/build/win32
   ```

   Note: *InstallDir* represents the location of your phoneME Feature software.

2. Ensure that all environment variables are set.

   For more information on environment variables, see Build Environment and Building a

PCSL Reference Port.

3. Ensure that a PCSL reference port is built for the Win32/i386 target platform.

   For more information on building PCSL, see Building a PCSL Reference Port.

4. Ensure that a CLDC reference port is built for the Win32/i386 target platform.

   For more information on building CLDC, see Building a CLDC Reference Port.

5. Mount the phoneME Feature software `midp` directory at the root level.

   For example, if you installed the phoneME Feature software in `C:\jwc1.1.3`, use the following command:

   ```
   mount C:\jwc1.1.3\midp /jwc1.1.3/midp
   ```

6. Use `make` to build a reference port of the phoneME Feature software.

   ```
   $ make VAR_IMPORT_FROM=Default.gmk all
   ```

   This builds the full phoneME Feature software bundle, including the example executables.

   Note: To build Javadoc documentation at the same time you build the phoneME Feature software default build configuration, enter the following command in Step 6:

   ```
   $ make VAR_IMPORT_FROM=Default.gmk docs_html all
   ```

Building phoneME Feature Software Documentation

You can create HTML documents from phoneME Feature software from porting interfaces. To do this for the phoneME Feature software, use the `make docs_html` build target.

Generating Javadoc Tool Documentation

Building Javadoc tool documentation is the same for both Linux and Windows platforms.

1. Change to the default build directory for your target platform.

```
$ cd InstallDir/midp/build/platform
```

2. Build Javadoc tool HTML documentation.

   ```
   $ make docs_html
   ```

   The generated HTML documents are put in the following directory:

   ```
   $PHONEME_OUTPUT_DIR/doc/javadoc/html
   ```

Viewing phoneME Feature Software Documents

Use any browser to display the Javadoc output file at the following URL:

```
file:///$PHONEME_OUTPUT_DIR/doc/javadoc/html/index.html
```

Using phoneME Feature Software

Once you build the phoneME Feature software for your target platform, it provides an environment in which MIDlet suites can be installed, listed, run, and uninstalled.

Installing a MIDlet Suite

For end users, the Over the Air (OTA) installer is used to install a MIDlet suite by clicking on the graphical link to a Java Application Descriptor (JAD) or Java Archive (JAR) file. For developers, a shell script is provided, called `installMidlet` that runs the OTA Installer from the command line. The `installMidlet` script is located in the directory `$MIDP_OUTPUT_DIR/bin/`*`cpu`*.

1. Change to the directory containing the `installMidlet` script.

   For example, on a Linux/ARM platform, change to the following directory:

   ```
   $ cd $MIDP_OUTPUT_DIR/bin/arm
   ```

2. To run the `installMidlet` script, enter the following in a command line:

   ```
   $ installMidlet url
   ```

The argument *url* indicates the URL of a JAD or a JAR file. The MIDlet suite is installed from this URL.

Note: The OTA installer used by the phoneME Feature software uses only the HTTP and HTTPS protocols to install MIDlet suites.

For more information on the OTA installer and `installMIDlet` subcommands, see the *Sun Java Wireless Client Build Guide*.

Using the *runMidlet* Command

The `runMidlet` application is built by default when you run the `make all` command to build the phoneME Feature software. The `runMidlet` script is located in the directory `$MIDP_OUTPUT_DIR/bin/`*`cpu`*.

Several arguments can be used with the `runMidlet` command, but this guide describes only two. For a complete list of arguments for `runMidlet`, see the *Sun Java Wireless Client Build Guide*.

1. Change to the directory that contains the `runMidlet` script.

   For example, on a Linux/ARM platform, change to the following directory:

   ```
   $ cd $MIDP_OUTPUT_DIR/bin/arm
   ```

2. Enter the following in a command line:

   ```
   $ runMidlet suiteNumber | suiteID
   ```

   The `runMidlet` command takes the argument *suiteNumber* | *suiteID*, which is the number or storage identifier given to the MIDlet suite when it was installed.

Removing an Installed MIDlet Suite

To remove one or all MIDlet suites installed in your phoneME Feature software environment, use the `removeMidlet` command. The `removeMidlet` script is located in the directory `$MIDP_OUTPUT_DIR/bin/`*`cpu`*.

1. Change to the directory that contains the `removeMidlet` script.

For example, on a Linux/ARM platform, change to the following directory:

```
$ cd $MIDP_OUTPUT_DIR/bin/arm
```

2. To run the `removeMIDlet` command, enter the following in a command line:

   ```
   $ removeMidlet ( suiteNumber | suiteID | all )
   ```

   The `removeMidlet` command takes the argument *suiteNumber* | *suiteID*, which is the number or storage identifier given to the MIDlet suite when it was installed.

   The argument `all` requests that all MIDlet suites be removed.

Default Build Configuration Options

This document describes the options used to build the PCSL and phoneME Feature software default configurations. It has the following sections:

The options described in this document are set when you run the build procedures described in Building a PCSL Reference Port and Building a phoneME Software Reference Port. They are presented here for reference only.

Note: For success with the instructions in this guide, do *not* change the default settings. For information on build configurations and changing default settings, see the *Sun Java Wireless Client Build Guide*.

Build Options for PCSL Software

Building PCSL software on your build platform (Linux or Windows) uses the default build options shown in TABLE A-1. Running the command `make all` in the PCSL build environment sets these options for you.

TABLE A-1PCSL Default Configuration Build Option Settings

Module Name	Value	Description
`FILE_MODULE`	`posix` (Linux) `win32` (Windows)	Use a `posix` file system on Linux. On Windows, use `win32`. Example: `FILE_MODULE=posix`
`MEMORY_MODULE`	`malloc`	Use standard C `malloc`. Example: `MEMORY_MODULE=malloc`
`USE_DATAGRAM`	`true`	Enable datagram APIs. Example: `USE_DATAGRAM=true`

NETWORK_MODULE	bsd/generic (Linux) winsock (Windows)	On Linux, use bsd/generic. On Windows, use winsock. Example: NETWORK_MODULE=bsd/generic
PRINT_MODULE	stdout	Print to stdout. Example: PRINT_MODULE=stdout
PCSL_CHUNKMEM_IMPL	pcsl_chunkheap	Build with the named file, which implements the public API pcsl_mem_*_chunk.
PCSL_CHUNKMEM_DIR	*InstallDir*/memory/heap	Build with the named directory, which holds the implementation of pcsl_mem_*_chunk.
USE_DEBUG	false	Build to optimize compilation. Do not include debugging information. Example: USE_DEBUG=false

---

Build Options for phoneME Feature Software

The default configuration file, Default.gmk, is a plain text file located in the default build directory of your build platform. For example, on a Linux platform, the Default.gmk file can be found in the following location:

InstallDir/midp/build/linux_fb_gcc

Note: *InstallDir* is the location of your phoneME Feature software.

Build Options for Core MIDP Package

TABLE A-2 describes the build options used to build core MIDP technology in the phoneME Feature software default configuration. For more information, see Building a phoneME Feature Reference Port.

For more information on additional core MIDP build options, see the *Sun Java Wireless Client Build Guide*.

TABLE A-2 Default Build Options for Core MIDP

Name	Value	Description

`TARGET_CPU`	`arm`
	(Linux)
	`i386`
	(Win32)
		Builds an implementation for the specified target platform.
		Example:
		`TARGET_CPU=arm`

`SUBSYSTEM_LCDUI_MODULES`	`chameleon`	Builds an implementation that uses adaptive user-interface technology. Example: `SUBSYSTEM_LCDUI_MODULES=chameleon`
`USE_MULTIPLE_ISOLATES`	`true`	Builds an implementation able to run more than one MIDlet at a time. Example: `USE_MULTIPLE_ISOLATES=true`
`USE_CLDC_11`	`true`	Builds an implementation compliant with the CLDC 1.1 Specification. Example: `USE_CLDC_11=true`

`MIDP_USE_ABB`	`true`	Builds an implementation using the MIDP Audio Building Block (ABB) sound capability. Example: `MIDP_USE_ABB=true`
`USE_JPEG`	`true`	Builds an implementation that uses the JPEG graphics recorder provided by the phoneME Feature software. Example: `USE_JPEG=true`
`JPEG_DIR`	(jpeg source)	Provides a pointer to the location of the `libjpeg` open source library. Example: `JPEG_DIR=`*InstallDir*`/libjpeg`
`USE_SSL`	`true`	Builds an implementation that includes an SSL library. Example: `USE_SSL=true` Note that this build flag does not affect security for OTA and SATSA. For that, see `USE_RESTRICTED_CRYPTO`.
`USE_RESTRICTED_CRYPTO`	`true`	Builds an implementation that includes ciphers and features that depend on ciphers. The features that depend on ciphers are: secure OTA, SecureConnection, HTTPS, and SATSA-CRYPTO (JSR 177). Example: `USE_RESTRICTED_CRYPTO=true`

RESTRICTED_CRYPTO_DIR	(crypto source)	Provides a pointer to the location of the restricted crypto source. Example: RESTRICTED_CRYPTO_DIR=*crypto_dir* where *crypto_dir* is the location of your restricted crypto source.
USE_BINARY_CRYPTO	false	Set this variable to true if you do not have restricted crypto source code. The build system will rebuild demos by linking with binary objects instead of attempting to compile source code.
USE_NATIVE_AMS	false	Builds an implementation that uses the AMS written in the Java programming language. Example: USE_NATIVE_AMS=false
USE_FIXED	false	Builds an implementation that uses the default resource policy, open-for-competition. Example: USE_FIXED=false
USE_DEBUG	false	Builds an optimized implementation without debugging enabled. Example: USE_DEBUG=false
USE_I3_TEST	false	Builds an implementation without unit tests enabled. Example: USE_I3_TEST=false

USE_JAVA_DEBUGGER	false	Disables Java platform debugger support, also known as KDWP. Example: USE_JAVA_DEBUGGER=false
USE_JAVA_PROFILER	false	Disables the profiler feature of the phoneME Feature software in CLDC. Example: USE_JAVA_PROFILER=false

Build Options for Optional Package JSRs

TABLE A-3 describes the options used to build optional package JSRs in the phoneME Feature software default configuration. For more information on each optional package and other build settings, see the *Sun Java Wireless Client Build Guide*.

For each optional package shown in TABLE A-3, it is necessary to set two environment variables: one to indicate that an optional package should be built; the other to point to the location where your optional package source files are installed.

For example, to build the Bluetooth (JSR 82) optional package, the following variables must be set:

- USE_JSR_82=true
- JSR_82_DIR=*InstallDir*/jsr82

TABLE A-3 Default Build Options for Optional Package JSRs

Name	Default Value	Description
USE_JSR_75	true	Builds an implementation of JSR 75 (Personal Information and File Management). Example: USE_JSR_75=true

`JSR_75_DIR`	(JSR 75 source)	Provides a pointer to the location of the JSR 75 source code. Example: `JSR_75_DIR=`*InstallDir*`/jsr75`
`USE_JSR_82`	`true`	Builds an implementation of JSR 82 (Bluetooth). Example: `USE_JSR_82=true`
`JSR_82_DIR`	(JSR 82 source)	Provides a pointer to the location of the JSR 82 source code. Example: `JSR_82_DIR=`*InstallDir*`/jsr82`
`USE_JSR_120`	`true`	Builds an implementation of JSR 120 (Java Wireless Messaging 1.0). Example: `USE_JSR_120=true` (Note: Setting `JSR_205_DIR` includes JSR 120.)
`USE_JSR_135`	`true`	Builds an implementation of JSR 135 (Mobile Media API). Example: `USE_JSR_135=true` (Note: To set the directory location for JSR 135 source code, use `JSR_234_DIR`.)
`USE_JSR_172`	`true`	Builds an implementation of JSR 172 (Java Web Services). Example: `USE_JSR_172=true`

`JSR_172_DIR`	(JSR 172 source)	Provides a pointer to the location of the JSR 172 source code. Example: `JSR_172_DIR=`*InstallDir*`/jsr172`
`USE_JSR_177`	`true`	Builds an implementation of JSR 177 (Java Security and Trust Services). Example: `USE_JSR_177=true`
`JSR_177_DIR`	(JSR 177 source)	Provides a pointer to the location of the JSR 177 source code. Example: `JSR_177_DIR=`*InstallDir*`/jsr177`
`USE_JSR_205`	`true`	Builds an implementation of JSR 205 (Java Wireless Messaging 2.0). Example: `USE_JSR_205=true`
`JSR_205_DIR`	(JSR 205 source)	Provides a pointer to the location of the JSR 205 source code. Example: `JSR_205_DIR=`*InstallDir*`/jsr205` (Note: Setting JSR_205_DIR includes JSR 120.)
`USE_JSR_226`	`true`	Builds an implementation of JSR 226 (Scalable 2D Vector Graphics). Example: `USE_JSR_226=true`

`JSR_226_DIR`	(JSR 226 source)	Provides a pointer to the location of the JSR 226 source code. Example: `JSR_226_DIR=`*InstallDir*`/jsr226`
`PISCES_DIR`	(path to 2D Renderer Subsystem)	Provides a pointer to the location of the 2D Renderer Subsystem. Example: `PISCES_DIR=`*InstallDir*`/pisces`
`jsr_234_DIR`	(JSR 135 source)	Provides a pointer to the location of the JSR 135 source code. Example: `JSR_234_DIR=`*InstallDir*`/jsr135`