Getting Started Guide

phoneME™ Feature, Version 1.0

November 2006

Please
Recycle

Contents

Contents iii

1. Introducing phoneME Feature Software 1

How to Use This Document 1

What's In This Release 2

What's Not in This Release 3

Getting More Information 3

2. The Build Environment 5

Overview 5

Required Software and Build Tools 6

Requirements for Building on a Linux Platform 6

Requirement for Building on a Windows Platform 6

Targets and Options 7

Setting Up Your Build Environment 8

Setting Up on a Linux Platform 8

Setting Variables on a Linux Platform 9

Setting Up on a Windows Platform 9

Setting Variables on a Windows Platform 10

3. Building a PCSL Reference Port 11

PCSL Environment Variables 12

- Setting Variables on a Linux Platform 12
- Setting Variables on a Windows Platform 13

Using PCSL `make` Files 14

PCSL Build Targets 14

Using the `all` Build Target 15

Building PCSL Software 16

- Building on a Linux Platform 16
- Building on a Windows Platform 17

Building PCSL Documentation 17

- Generating Doxygen for a Linux Platform 18
- Generating Doxygen for a Windows Platform 18

Viewing PCSL Documents 18

4. Building an OPEN_SOURCE_VM Reference Port 19

Overview 19

Setting Up Your Build Environment 20

Setting Up on a Linux Platform 20

Setting OPEN_SOURCE_VM Variables 21

Setting Up on a Windows Platform 21

Setting OPEN_SOURCE_VM Variables 22

Mapping Between Software Components 22

Building OPEN_SOURCE_VM Software 22

- Building on the Linux Platform 23
- Building on the Windows Platform 23

Building OPEN_SOURCE_VM Documentation 23

- Generating Javadoc for a Linux Platform 23
- Generating Javadoc for a Windows Platform 24

Viewing phoneME Feature Documents 24

Running OPEN_SOURCE_VM Software 25

- Running on the Linux Platform 25
- Running on the Windows Platform 25

5. Building the phoneME Feature Software 27

Using phoneME Feature `make` Files 27

Default Build Configurations 28

phoneME Feature Build Targets 28

Using the `all` Build Target 29

phoneME Feature Reference Port 30

- Building phoneME Feature on a Linux Platform 31
- Building phoneME Feature on a Windows Platform 32

Building phoneME Feature Documentation 33

- Generating Javadoc for a Linux Platform 33
- Generating Javadoc for a Windows Platform 33

Viewing phoneME Feature Documents 34

Using phoneME Feature Software 34

- Installing a MIDlet Suite 34
- Listing an Installed MIDlet Suite 35

Running an Installed MIDlet Suite 36

- Removing an Installed MIDlet Suite 37

Index 39

CHAPTER 1

Introducing phoneME Feature Software

The phoneME™ Feature software is an Open Source Software (OSS) community version of Sun Microsystems' Sun Java™ Wireless Client software commercial product. Although primarily created from the same shared code base, Sun Microsystems' commercial product may not fully and accurately represent the OSS source base, due to licensing and other legal restrictions.

Sun Microsystems provides a complete set of Sun Java Wireless Client software documentation, which is available for your review. For a complete list of documents in the Sun Java Wireless Client software documentation set, see:

http://java.sun.com/javame/reference/docs/sjwc-1.1.3-web/index.html

How to Use This Document

This *Getting Started Guide* provides basic information that allows you to take the following steps:

- Setup and configure your Linux/ARM or Win32/i386 build environment
- Build a default implementation of the PCSL libraries, OPEN_SOURCE_VM, and phoneME Feature software binaries
- Run the phoneME Feature software binary implementation on your default platform
- Find additional information about phoneME Feature software

This guide is not exhaustive. It 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. Many build options and additional features are available that are not covered in this guide.

For complete instructions on how to customize and build non-default versions of the PCSL libraries, OPEN_SOURCE_VM, and phoneME Feature software, see the *Sun Java Wireless Client Build Guide*, at the following location:

http://java.sun.com/javame/reference/docs/sjwc-1.1.3-web/index.html

This guide assumes that the phoneME Feature software is installed on a system that meets the hardware and software requirements described inChapter 2.

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)
- Enhanced multitasking capabilities and support for multiple, simultaneous applications (MIDlets)
- Support for several Java Specification Request (JSR) optional packages:
  - Personal Information and File Management (JSR 75)
  - Java 2 Platform, Micro Edition (J2ME™ platform) Web Services (JSR 172)
  - Security and Trust Services API (JSR 177)
  - Wireless Messaging 2.0 (JSR 205)
  - Scalable 2D Vector Graphics (JSR 226)
- Customizable user interface (skin) via Chameleon Adaptive User Interface Technology (AUIT) and XML
- Native Application Management API
- Native Resource Management API
- Separate bundle for cryptography source

For more information, see the Sun Java Wireless Client software commercial documentation set, at the following location:

http://java.sun.com/javame/reference/docs/sjwc-1.1.3-web/index.html

What's Not in This Release

Due to licensing and other legal restrictions, the following software components are not included in the phoneME Feature software release:

- Bluetooth (JSR 82)
- Mobile Media API (JSR 135)
- Mobile 3D Graphics (JSR 184)

To work with these capabilities of phoneME Feature, you must acquire an appropriate implementation of these JSRs from outside the phoneME Feature source base.

Getting More Information

For more information about the phoneME Feature software code base, including porting instructions, supporting additional platforms, customizing the Adaptive User Interface, and multitasking considerations, see the Sun Java Wireless Client software documentation set. This set can be found in the following location:

http://java.sun.com/javame/reference/docs/sjwc-1.1.3-web/index.html

The Sun Java Wireless Client software documentation set describes Sun Microsystems' commercial version of the phoneME Feature software. TABLE 1-1 provides a complete list of the documents found in this set.

TABLE 1-1 Sun Java Wireless Client Software Documentation Set

Title	Description
Sun Java Wireless Client Release Notes	Provides important information on the latest release of the Sun Java Wireless Client software.
Sun Java Wireless Client Build Guide	Provides complete instructions for building the Sun Java Wireless Client software.
Sun Java Wireless Client Porting Guide	Describes the considerations and procedures used for porting the Sun Java Wireless Client software.
Sun Java Wireless Client Skin Author's Guide	Describes how to customize the default adaptive user interface (skin) for the Sun Java Wireless Client software.

TABLE 1-1 Sun Java Wireless Client Software Documentation Set *(Continued)*

Title	Description
Sun Java Wireless Client Tools Guide	Describes `MEKeyTool`, `JADTool`, `JARTool`, and other tools used with the Sun Java Wireless Client software.
Sun Java Wireless Client Multitasking Guide	Describes issues you must consider when using multitasking features in the Sun Java Wireless Client.
Sun Java Wireless Client Multitasking Test Suite Guide	Describes MT-QTS, the test suite used to test multitasking in the Sun Java Wireless Client software.

CHAPTER 2

The Build Environment

This *Getting Started Guide* describes only those steps needed to build a default phoneME Feature implementation for the following platforms:

- Linux/ARM
- Win32/i386

Building on other platforms is possible but not supported. For more information on how to build phoneME Feature software on other platforms, see the *Sun Java Wireless Client Build Guide* in the following location:

http://java.sun.com/javame/reference/docs/sjwc-1.1.3-web/index.html

Overview

To properly build the phoneME Feature software, you must do the following:

- Build an implementation of the Portable Common Services Libraries (PCSL)
- Build an implementation of the OPEN_SOURCE_VM virtual machine software
- Build an implementation of the phoneME Feature software

Note – The phoneME Feature software includes a number of optional Java Specification Requests (JSRs), as described in "What's In This Release" on page 2. These optional packages are not built by default. For instructions on how to include them in your build, see the *Sun Java Wireless Client Build Guide* at http://java.sun.com/javame/reference/docs/sjwc-1.1.3-web/index.html

---

Required Software and Build Tools

The following are the supported platforms for building an implementation of PCSL, OPEN_SOURCE_VM software, and phoneME Feature software:

- Linux/ARM
- Win32/i386

The following are the supported platforms for running an implementation of the phoneME Feature software:

- Texas Instruments P2SAMPLE64-V6 with Linux/ARM
- Windows/i386 (which runs the product in emulation mode)

Requirements for Building on a Linux Platform

A Linux/ARM system must meet the following requirements:

- Red Hat Linux distribution version 7.2 - 9.0 for the Sun Java Desktop System 1.1 and 1.2
- Java 2 Platform, Standard Edition (J2SE™) Development Kit (JDK™) version 1.4.2
- GNU Make version 3.79.1 or later
- OPEN_SOURCE_VM
- GNU Cross Compiler (GCC) 3.3.1 or later (3.4.6 or later for the OPEN_SOURCE_VM software)
- Doxygen version 1.4.4

Requirement for Building on a Windows Platform

A Win32/i386 platform must meet the following requirements:

- Windows 2000 or Windows XP
- Java 2 Platform, Standard Edition (J2SE) Development Kit (JDK) version 1.4.2
- GNU Make version 3.79.1 or later
- OPEN_SOURCE_VM
- 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 1.1.1, an implementation of Cygwin for building Java ME technology
- Windows MKS Toolkit (optionally, as an alternative to Cyg4Me. If installing MKS Toolkit, GNU Make is included.)

Note – If you are building OPEN_SOURCE_VM on a Windows platform with output targeted for the ARM platform, you must use Microsoft eMbedded Visual C++, version 3.0 or later.

Targets and Options

The phoneME Feature software and its supporting components (PCSL and OPEN_SOURCE_VM) are built using the makefiles and source files provided for each component.

In this *Getting Started Guide*, each component is built using only the default settings provided by the component. Therefore, to properly build a component such as the phoneME Feature software, it is only necessary to properly set environment variables (as described in this and subsequent chapters) and enter the `make` command, as shown here:

```
$ make all
```

Note – When building the OPEN_SOURCE_VM software, the command used is:
`$ gnumake`

For a complete description of the targets and configuration options provided for building PCSL and phoneME Feature software, see the *Sun Java Wireless Client Build Guide* at the following location:

http://java.sun.com/javame/reference/docs/sjwc-1.1.3-web/index.html

For a complete description of the targets and configuration options provided for building the OPEN_SOURCE_VM software, see the *CLDC HotSpot Implementation Build Guide* at the following location:

http://java.sun.com/javame/reference/docs/cldc-hi-1.1.3-web/doc/build/pdf/CLDC-Hotspot-Build.pdf

Setting Up Your Build Environment

Preparing your environment to build the phoneME Feature software requires you to set environment variables for your platform (on Linux and Windows), as well as take other steps (on Windows).

Other environment variables may also need to be set, which are specific to a particular component being built. Environment variables for specific components are described in the chapter for that component. For example, PCSL-specific environment variables are described in Chapter 3.

If you are building the phoneME Feature software on a Linux platform, see "Setting Up on a Linux Platform" on page 8. If you are building the phoneME Feature software on a Windows platform, see "Setting Up on a Windows Platform" on page 9.

Setting Up on a Linux Platform

To properly build the phoneME Feature software on a Linux platform, you must set the environment variables shown in TABLE 2-1. Once these variables are set, no other environment set up is needed.

TABLE 2-1 Linux Platform Environment Variables

Name	Description
`JDK_DIR`	Directory that contains the JDK release. For example, `/usr/java/j2sdk1.4.1_07`.
`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.

Setting Variables on a Linux Platform

Note – The following examples assume the use of the Bash shell on the Linux platform. If you are using some other shell than Bash, set the variables for the shell you are using.

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

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

To set `MIDP_DIR` on a `make` command line:

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

Setting Up on a Windows Platform

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

- Installing Cyg4Me
- Setting Variables on a Windows Platform
- Updating Your PATH Environment Variable

▼ Installing Cyg4Me

Before you build phoneME Feature on a Windows platform for the first time, download and install Cyg4Me, version 1.1.1. 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 Cyg4Me from `ftp://ftp.sunfreeware.com/pub/freeware/contributions/cygwin/cyg4me1_1_full.zip`.**

   The file contains the executables and libraries required to build the products, plus the source code as required by the General Public License.

2. **Unpack the file into a directory.**

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

   Every time you build phoneME Feature, ensure that the Cyg4Me `bin` directory is the only version of Cygwin on your path. See "Updating Your PATH Environment Variable" on page 10 for instructions.

Setting Variables on a Windows Platform

To properly 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/j2sdk1.4.1_07`.
`MIDP_DIR`	Top level MIDP directory, usually *InstallDir*`/jwc/midp`
`MIDP_OUTPUT_DIR`	Location where output from phoneME Feature build is stored.

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

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

To set `MIDP_DIR` on a `make` command line:

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

▼ Updating Your PATH Environment Variable

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

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

   For example, you could set your `PATH` environment variable 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.

CHAPTER 3

Building a PCSL Reference Port

This chapter contains instructions for building a PCSL reference port. The phoneME Feature software uses the resulting libraries when building an implementation.

PCSL contains the following individual services:

- The `file` Service
- The `memory` Service
- The `network` Service
- The `print` Service

Note – Although each PCSL service can be individually built, this *Getting Started Guide* describes only how to build the full PCSL.

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*, at the following location:

http://java.sun.com/javame/reference/docs/sjwc-1.1.3-web/index.html

PCSL Environment Variables

PCSL has two environment variables that can be set. The first is mandatory; the second is optional, 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*.
- `PCSL_OUTPUT_DIR` - specifies into which directory the build system puts its output (e.g., object files and libraries).

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

PCSL environment variables can be set as a system variable, or on the command line when you run `make`.

▼ Setting Variables on a Linux Platform

Note – The following examples assume the use of the Bash shell on the Linux platform. If you are using some other shell than Bash, set the variables for the shell you are using.

---

1. **Set the `PCSL_PLATFORM` environment variable.**

   To set `PCSL_PLATFORM` as a system variable:

   ```
   $ export PCSL_PLATFORM=linux_arm_gcc
   ```

   To set `PCSL_PLATFORM` on a `make` command line:

   ```
   $ make PCSL_PLATFORM=linux_arm_gcc
   ```

2. **(Optional) Set the `PCSL_OUTPUT_DIR` environment variable.**

   To set `PCSL_OUTPUT` as an system variable:

   ```
   $ export PCSL_OUTPUT_DIR=$HOME/pcsl_output
   ```

   To set `PCSL_OUTPUT` on a `make` command line:

   ```
   $ make PCSL_OUTPUT_DIR=$HOME/pcsl_output
   ```

Note – Although you can build the PCSL reference port without setting PCSL_OUTPUT_DIR, you *must* set it when you build phoneME Feature. Its value must be the same for both builds.

If you do not set a value for PCSL_OUTPUT, the build system creates a default output directory at the top of the PCSL workspace. For example, if the directory *InstallDir*/pcsl is the top of the workspace, then the default PCSL_OUTPUT_DIR is *InstallDir*/pcsl/output.

▼ Setting Variables on a Windows Platform

1. **Set the** PCSL_PLATFORM **environment variable.**

   To set PCSL_PLATFORM as a system variable:

   ```
   $ set PCSL_PLATFORM=win32_i386_vc
   ```

   To set PCSL_PLATFORM on a make command line:

   ```
   $ make PCSL_PLATFORM=win32_i386_vc
   ```

2. **(Optional) Set the** PCSL_OUTPUT_DIR **environment variable.**

   To set PCSL_OUTPUT as an system variable:

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

   To set PCSL_OUTPUT on a make command line:

   ```
   $ make PCSL_OUTPUT_DIR=C:/my_output/pcsl/output
   ```

Note – Although you can build the PCSL reference port without setting PCSL_OUTPUT_DIR, you *must* set it when you build phoneME Feature. Its value must be the same for both builds.

If you do not set a value for PCSL_OUTPUT, the build system creates a default output directory at the top of the PCSL workspace. For example, if the directory *InstallDir*/pcsl is the top of the workspace, then the default PCSL_OUTPUT_DIR is *InstallDir*/pcsl/output.

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 following directory:

InstallDir`/pcsl/makefiles/platforms`

In the path above, *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. Although several build targets are available, this *Getting Started Guide* uses only two, as shown in TABLE 3-1.

TABLE 3-1 PCSL Build Targets

Name	Description
`all` (default)	Builds 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" on page 17.

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

Using the `all` Build Target

The default build target for every makefile is `all`. Running the `make all` command builds the PCSL software development environment, the libraries, the include files, and the default modules, as shown in TABLE 3-2.

TABLE 3-2 PCSL Default Modules

Module Name	Value	Description
`FILE_MODULE`	`posix` (default)	Use a POSIX file system. Example: `FILE_MODULE=posix`
`MEMORY_MODULE`	`malloc` (default)	Use standard C `malloc`. Example: `MEMORY_MODULE=malloc`
`USE_DATAGRAM`	`true`	Enable datagram APIs. Example: `USE_DATAGRAM=true`
`NETWORK_MODULE`	`bsd/qte` (default)	Use Linux Qt Embedded implementation. Example: `NETWORK_MODULE=bsd/qte`
`PRINT_MODULE`	`stdout` (default)	Print to `stdout`. Example: `PRINT_MODULE=stdout`
`PCSL_CHUNKMEM_IMPL`	`pcsl_chunkheap` (default)	Build with the named file, which implements the public API `pcsl_mem_*_chunk`.
`PCSL_CHUNKMEM_DIR`	*InstallDir*`/memory/heap` (default)	Build with the named directory, which holds the implementation of `pcsl_mem_*_chunk`.
`USE_DEBUG`	`false` (default)	Build to optimize compilation. Do not include debugging information. Example: `USE_DEBUG=false`

Building PCSL Software

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

▼ Building on a Linux Platform

1. **Change to the PCSL directory.**

   `$ cd` *InstallDir*`/pcsl`

2. **Run the** `make all` **command.**

   If you have already set the PCSL environment variables, type:

   ```
   $ make all
   ```

   If you have not yet set the PCSL environment variables, type:

   ```
   $ make PCSL_PLATFORM=linux_arm_gcc all
   ```

Note – To build the phoneME Feature software for the Linux/ARM platform, you must build PCSL for both Linux/i386 and Linux/ARM.

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

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

Other directories are also created, but for software development purposes, you only need the `lib` and `inc` directories. For a complete list of created directories, see the *Sun Java Wireless Client Build Guide*, at the following location:

http://java.sun.com/javame/reference/docs/sjwc-1.1.3-web/index.html

▼ 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 all` command.**

   If you have already set the PCSL environment variables, type:

   ```
   $ make all
   ```

   If you have not yet set the PCSL environment variables, type:

   ```
   $ make PCSL_PLATFORM=win32_i386_vc all
   ```

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

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

   Other directories are also created, but for software development purposes, you only need the `lib` and `inc` directories. For a complete list of created directories, see the *Sun Java Wireless Client Build Guide*, at the following location:

   http://java.sun.com/javame/reference/docs/sjwc-1.1.3-web/index.html

Building PCSL Documentation

If you installed the Doxygen tool (from `http://www.doxygen.org`), you can create HTML documents from high-level public and low-level 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 – When building the documentation, you must set the `PCSL_PLATFORM` environment variable. Setting the `PCSL_OUTPUT_DIR` environment variable is optional. See "PCSL Environment Variables" on page 12 for more information.

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 for a Linux Platform

1. **Go to the PCSL directory:**

   `$ cd` *InstallDir*`/pcsl`

2. **Run the** `make doc` **command:**

   If you have already set the PCSL environment variables, type:

   ```
   $ make doc
   ```

   If you have not yet set the PCSL environment variables, type:

   ```
   $ make PCSL_PLATFORM=linux_arm_gcc doc
   ```

▼ Generating Doxygen for a Windows Platform

1. **Go to the PCSL directory:**

   `$ cd` *InstallDir*`/pcsl`

2. **Run the** `make doc` **command:**

   If you have already set the PCSL environment variables, type:

   ```
   $ make doc
   ```

   If you have not yet set the PCSL environment variables, type:

   ```
   $ make PCSL_PLATFORM=win32_i386_vc doc
   ```

Viewing PCSL Documents

Use any browser to display the Doxygen output file at the URL `file:///$PCSL_OUTPUT_DIR/doc/doxygen/html/index.html`.

(NOTE: need the correct URL here)

CHAPTER 4

Building an OPEN_SOURCE_VM Reference Port

The OPEN_SOURCE_VM software is an Open Source Software (OSS) community implementation of Sun Microsystems' Connected Limited Device Configuration HotSpot™ Implementation virtual machine (CLDC HotSpot Implementation). CLDC HotSpot Implementation is a high-performance virtual machine that can be used as an execution engine for the Connected Limited Device Configuration platform.

Although primarily created from the same shared code base, Sun Microsystems' commercial product may not fully and accurately represent the OSS source base, due to licensing and other legal restrictions.

Sun Microsystems provides a complete set of CLDC HotSpot Implementation software documentation, which is available for your review. A complete list of CLDC HotSpot Implementation software documentation is available at the following location:

http://java.sun.com/javame/reference/apis.jsp

Overview

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

In a development environment, building and running the OPEN_SOURCE_VM software would be just one part in a more extended process of building, porting, modifying, rebuilding, and porting again, until the OPEN_SOURCE_VM software was fully customized for your specific platform or device.

For more information on how to build, customize, and port the OPEN_SOURCE_VM software, see the *CLDC HotSpot Implementation* documentation set, at the following location:

http://java.sun.com/javame/reference/apis.jsp

Setting Up Your Build Environment

To properly build on the Win32/i386 and Linux/ARM platforms, only minimal setup of your build environment is needed, as shown below.

Setting Up on a Linux Platform

To properly build the OPEN_SOURCE_VM software on a Linux platform, you must set the environment variables shown in TABLE 4-1. Once these variables are set, no other environment set up is needed.

TABLE 4-1 Linux Platform Environment Variables

Name	Description
`JVMWorkSpace`	The location of your OPEN_SOURCE_VM software source code workspace.
`JVMBuildSpace`	The location of your OPEN_SOURCE_VM software build output.
`PATH`	Must be set to include all 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`.

Setting OPEN_SOURCE_VM Variables

An OPEN_SOURCE_VM variable can be set on a Linux platform in two ways:

- As a system variable
- On the `gnumake` command line

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

```
$ setenv JVMWorksSpace InstallDir/linux
```

To set `JVMWorkSpace` on a `gnumake` command line:

```
$ gnumake JVMWorkSpace=InstallDir/linux
```

Setting Up on a Windows Platform

To properly build the OPEN_SOURCE_VM software on a Windows platform, you must set the environment variables shown in TABLE 4-2. Once these variables are set, no other environment setup is needed.

TABLE 4-2 Windows Platform Environment Variables

Name	Description
`JVMWorkSpace`	The location of your OPEN_SOURCE_VM software source code workspace.
`JVMBuildSpace`	The location of your OPEN_SOURCE_VM software build output.
`PATH`	Must be set to include all 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`.

Setting OPEN_SOURCE_VM Variables

An OPEN_SOURCE_VM variable can be set on a Windows platform in two ways:

- As a system variable
- On the `gnumake` command line

For example. to set `JVMWorkSpace` as a system variable:

```
$ set JVMWorkSpace=win32
```

To set `JVMWorkSpace` on a `gnumake` command line:

```
$ gnumake JVMWorkSpace=win32
```

Mapping Between Software Components

If you choose not to build the default implementations and instead, decide to modify build configurations or option settings for either the OPEN_SOURCE_VM or phoneME Feature software, you must do so carefully. Settings used in one component *must* match the settings in the other component.

For more information on mappings between the OPEN_SOURCE_VM and phoneME Feature software components, see the *Sun Java Wireless Client Build Guide*, at the following location:

http://java.sun.com/javame/reference/docs/sjwc-1.1.3-web/index.html

Building OPEN_SOURCE_VM Software

The default OPEN_SOURCE_VM software build mode is *Debug*. It includes all the possible runtime assertions, debug and test code. This build version executes relatively slowly, but is valuable for testing and debugging the system. An example of a possible runtime assertion is checking on each bytecode dispatch.

Building a default version of the OPEN_SOURCE_VM software creates a debuggable version of the OPEN_SOURCE_VM software executable (`cldc_hi.exe`). (NOTE: is this the correct file name?)

Other build modes are available. For more information, see The *CLDC HotSpot Implementation Build Guide*, at the following location:

http://java.sun.com/javame/reference/docs/cldc-hi-1.1.3-web/doc/build/pdf/CLDC-Hotspot-Build.pdf

After building the OPEN_SOURCE_VM system, follow the instructions in "Running OPEN_SOURCE_VM Software" on page 25" to run it.

▼ Building on the Linux Platform

1. **Change directory to** `$JVMWorkSpace\build\linux_arm` **and run the** `gnumake` **command. For example,**

   ```
   $ gnumake debug
   ```

   Output is generated under `$JVMBuildSpace\build\linux_arm\target\bin`

▼ Building on the Windows Platform

1. **Change directory to** `%JVMWorkSpace%\build\win32_i386` **and run the** `gnumake` **command. For example,**

   ```
   $ gnumake debug
   ```

   Output is generated under`%JVMBuildSpace%\build\win32_i386\target\bin`

Building OPEN_SOURCE_VM Documentation

You can create HTML documents from OPEN_SOURCE_VM high-level public and low-level porting interfaces. To do this for the OPEN_SOURCE_VM software, use the `gnumake docs_html` build target.

▼ Generating Javadoc for a Linux Platform

1. **Go to the default build directory for the Linux/ARM platform.**

   ```
   $ cd $JVMWorkSpace/build/linux
   ```

   (NOTE: need correct path here.)

2. **Build Javadoc HTML documentation.**

   ```
   $ gnumake docs_html
   ```

The generated HTML documents are put in the directory `$JVMBuildSpace/doc/javadoc/html`.

(NOTE: need correct path here.)

▼ Generating Javadoc for a Windows Platform

1. **Go to the default build directory for the Linux/ARM platform.**

   ```
   $ cd %JVMWorkSpace%/build/linux
   ```

   (NOTE: need correct path here.)

2. **Build Javadoc HTML documentation.**

   ```
   $ gnumake docs_html
   ```

   The generated HTML documents are put in the directory `%JVMBuildSpace%/doc/javadoc/html`.

   (NOTE: need correct path here.)

Viewing phoneME Feature Documents

Use any browser to display the Javadoc output file at the URL `file:///$JVMBuildSpace/doc/javadoc/html/index.html`.

(NOTE: need correct path here.)

Running OPEN_SOURCE_VM Software

Once you have built a debug version of OPEN_SOURCE_VM, 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 which platform (operating system and processor) you have built for.

▼ Running on the Linux Platform

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

   ```
   $ cd $JVMBuildSpace\linux_arm
   ```

2. **Enter the following command:**

   ```
   $ bin\cldc_hi_g -classpath C:<location of compiled Java applications>\
   classes <classname>
   ```

   Many more command options are available to run the OPEN_SOURCE_VM software. For more information, see the *CLDC HotSpot Implementation Build Guide*, at the following location:

   http://java.sun.com/javame/reference/docs/cldc-hi-1.1.3-web/doc/build/pdf/CLDC-Hotspot-Build.pdf

▼ Running on the Windows Platform

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

   ```
   $ cd %JVMBuildSpace%\win32_i386
   ```

2. **Enter the following command:**

   ```
   $ bin\cldc_hi_g -classpath C:<location of compiled Java applications>\
   classes <classname>
   ```

   Many more command options are available to run the OPEN_SOURCE_VM software. For more information, see the *CLDC HotSpot Implementation Build Guide*, at the following location:

   http://java.sun.com/javame/reference/docs/cldc-hi-1.1.3-web/doc/build/pdf/CLDC-Hotspot-Build.pdf

CHAPTER 5

Building the phoneME Feature Software

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

This chapter has instructions for building a default configuration of the phoneME Feature software. For proper building and performance, your system must meet the requirements provided in the *phoneME Feature Release Notes.*

Using phoneME Feature `make` Files

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

The phoneME Feature software uses a top-level, platform-specific makefile, which is tailored to the target platform's operating system, CPU, and the compiler used to build the phoneME Feature implementation. For example, the makefile for the Windows implementation of phoneME Feature is called `win32_i386_vc.gmk`.

The *`OS_CPU_COMPILER`*`.gmk` file is located in the following directory:

InstallDir`/midp/build`

In the path above, *InstallDir* is the location of your phoneME Feature software.

Default Build Configurations

Running the `make` command builds a default build configuration for your target platform, which 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.

Note – Many build configurations are possible for phoneME Feature software on both the Linux/ARM and Win32/i386 platforms. However, this *Getting Started Guide* only provides instructions for building the default configuration for each platform. For information on how to build other configurations for other platforms, see the *Sun Java Wireless Client Build Guide*.

On the Linux/ARM platform (e.g., the P2AMPLE64-V6 board), the default build configuration uses QT graphics (qt), with Adaptive User Interface Technology (chameleon), and multitasking (mvm), as shown here:

- `linux_arm_qt_chameleon_mvm`

On the Win32/i386 platform, the default build configuration is:

- `win32_i386`

phoneME Feature Build Targets

The phoneME Feature software build system provides several `make` targets that enable you to build sections of the phoneME Feature reference port. Although several build targets are available, this *Getting Started Guide* uses only two, as shown in TABLE 5-1.

TABLE 5-1 phoneME Feature Software Build Targets

Target	Description
`all` (default)	Creates the executables, classes, and tools for an implementation of the phoneME Feature software.
`docs_html`	Generates Porting API documentation.

For more information on phoneME Feature build targets, see the *Sun Java Wireless Client Build Guide*, at the following location:

http://java.sun.com/javame/reference/docs/sjwc-1.1.3-web/index.html

Using the `all` Build Target

The default build target for every makefile is `all`. Running the top-level `make all` command builds the phoneME Feature software development environment, the libraries, the include files, and the default modules, as shown in TABLE 5-2.

TABLE 5-2 Default Build Settings and Values

Name	Value	Description
`USE_CLDC_11`	`true` (default)	Builds an implementation compliant with the CLDC 1.1 Specification. Example: `USE_CLDC_11=true`
`SUBSYSTEM_LCDUI_MODULES`	`chameleon` (default)	Builds an implementation that uses adaptive user-interface technology. Example: `SUBSYSTEM_LCDUI_MODULES=chameleon`
`USE_NATIVE_AMS`	`false` (default)	Builds an implementation that uses the AMS written in the Java programming language. Example: `USE_NATIVE_AMS=false`
`USE_MULTIPLE_ISOLATES`	`false` (default)	Builds an implementation that is capable of running more than one MIDlet at a time. Example: `USE_MULTIPLE_ISOLATES=false`
`USE_FIXED`	`false` (default)	Builds an implementation that uses the default resource policy, open-for-competition. Example: `USE_FIXED=false`
`USE_SSL`	`false` (default)	Builds an implementation that does not include an SSL library. Example: `USE_SSL=false` Note that this build flag does not affect security for OTA and SATSA. For that, see `USE_RESTRICTED_CRYPTO`.
`USE_RESTRICTED_CRYPTO`	`false` (default)	Builds an implementation that does not include 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=false`

TABLE 5-2 Default Build Settings and Values

Name	Value	Description
USE_BINARY_CRYPTO	false (default)	Set this variable to true if you do not have the restricted_crypto source code. The build system will rebuild demos by linking with binary objects instead of attempting to compile source code.
USE_SERVER_SOCKET	true (default)	Builds an implementation with server socket protocol support. Example: USE_SERVER_SOCKET=true
USE_DEBUG	false (default)	Builds an optimized implementation without debugging enabled. Example: USE_DEBUG=false
USE_I3_TEST	false (default)	Builds an implementation without unit tests enabled. Example: USE_I3_TEST=false
USE_JAVA_DEBUGGER	false (default)	Disables Java platform debugger support, also known as KDWP. Example: USE_JAVA_DEBUGGER=false
USE_JAVA_PROFILER	false (default)	Disables the profiler feature of the Sun Java Wireless Toolkit in the OPEN_SOURCE_VM. Example: USE_JAVA_PROFILER=false

phoneME Feature Reference Port

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.

Note – Although you can build the PCSL reference port without setting the variable PCSL_OUTPUT_DIR, you *must* set it when you build the phoneME Feature software. Its value must be the same for both builds.

▼ Building phoneME Feature on a Linux Platform

1. **Acquire Monta Vista developer tools.**

   To properly build phoneME Feature software on the Linux ARM (P2AMPLE64-V6) platform, you must acquire the MontaVista CEE 3.1 SDK developer tools, provided by MontaVista. Instructions for use of these tools is provided by MontaVista.

2. **Set** `GNU_TOOLS_DIR` **Variable.**

   Once you have acquired the MontaVista CEE 3.1 SDK developer tools, for proper cross-compilation, you must set the `GNU_TOOLS_DIR` environment variable. For example:

   ```
   export GNU_TOOLS_DIR=
   /opt/montavista/cee/devkit/arm/v4t_le/arm4vtl-hardhat-li
   ```

3. **Verify that the environment variables are set.**

   See Chapter 2 and Chapter 3.

4. **Verify that a PCSL reference port is built for the target platform.**

   To build phoneME Feature for the ARM platform, you must first build PCSL for both i386 and ARM.

   If you need to build PCSL, see Chapter 3 for detailed instructions.

5. **Verify that a OPEN_SOURCE_VM reference port is built for the target platform.**

   To build phoneME Feature software for the ARM platform, you must first build OPEN_SOURCE_VM for both i386 and ARM.

   If you need to build the OPEN_SOURCE_VM, see Chapter 4 for detailed instructions.

6. **Go to the default phoneME Feature build directory for the Linux/ARM platform.**

   $ cd *InstallDir*`/midp/build/linux_arm_gcc`

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

   ---

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

   For example:

   ```
   $ make all
   ```

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

Note – Building the phoneME Feature software Javadoc documentation is something that can be done at any time, as described in "Building phoneME Feature Documentation" on page 33. However, it can also be built at the same time as your phoneME Feature software reference port, by typing:

```
$ make all docs_html
```

▼ Building phoneME Feature on a Windows Platform

1. **Verify that the environment variables are set.**

   See Chapter 2 and Chapter 3.

2. **Verify that a PCSL reference port is built for the target platform.**

   If you need to build PCSL, see Chapter 3 for detailed instructions.

3. **Verify that a OPEN_SOURCE_VM reference port is built for the target platform.**

   If you need to build the OPEN_SOURCE_VM, see Chapter 4 for detailed instructions.

4. **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
   ```

5. **Go to the default phoneME Feature build directory for the Windows/i386 platform.**

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

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

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

   For example:

   ```
   $ make all
   ```

   Note – When you provide options to the `make` command on a Windows platform, use a slash (/) as the separator in the file names.

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

Note – Building the phoneME Feature software Javadoc™ documentation is something that can be done at any time, as described in "Building phoneME Feature Documentation" on page 33. However, it can also be built at the same time as your phoneME Feature software reference port, by typing:

```
$ make all docs_html
```

Building phoneME Feature Documentation

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

▼ Generating Javadoc for a Linux Platform

1. **Go to the default build directory for the Linux/ARM platform.**

   ```
   $ cd InstallDir/midp/build/linux_arm_gcc
   ```

2. **Build Javadoc HTML documentation.**

   ```
   $ make docs_html
   ```

   The generated HTML documents are put in the directory `$PHONEME_OUTPUT_DIR/doc/javadoc/html`.

   (NOTE: need correct path here.)

▼ Generating Javadoc for a Windows Platform

1. **Go to the default build directory for the Linux/ARM platform.**

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

2. **Build Javadoc HTML documentation.**

   ```
   $ make docs_html
   ```

The generated HTML documents are put in the directory `$PHONEME_OUTPUT_DIR/doc/javadoc/html`.

(NOTE: need correct path here.)

Viewing phoneME Feature Documents

Use any browser to display the Javadoc output file at the URL `file:///$PHONEME_OUTPUT_DIR/doc/javadoc/html/index.html`.

(NOTE: need correct URL here.)

Using phoneME Feature Software

One you have built 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 JAD or 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 `installMidlet` command takes the following argument:

   - *url*

     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*.

▼ Listing an Installed MIDlet Suite

If one or more MIDlet suites are installed in your phoneME Feature software environment, use the `listMIDlet` command to get more information about them. The `listMidlet` script is located in the directory `$MIDP_OUTPUT_DIR/bin/`*cpu*.

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

1. **Change to the directory that contains the `listMidlet` script.**

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

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

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

   ```
   $ listMidlets
   ```

Note – There are no arguments to the `listMIDlets` command.

If one or more MIDlet suites are installed, the following information is displayed.

```
[1]
  Name: SunSamples - Games
  Vendor: Sun Microsystems, Inc.
  Version: 1.0.3
  Description: Sample suite of games for the MIDP.
  Authorized by: C=US;O=Root Test CA
  Security Domain: operator
  Suite ID:
#Sun%0020#Microsystems%002c%0020#Inc%002e_#Sun#Samples%0020%002d%00
20#Games_
  JAD URL: http://host.sun.com/games.jad
  JAR URL: http://host.sun.com/games.jar
  Size: 34K
```

Note – The number shown above in square brackets is the MIDlet suite number.

The information shown here is useful if you want to run or uninstall an existing MIDlet suite.

Running an Installed MIDlet Suite

There are two commands available to run a MIDlet from an installed MIDlet suite:

- `runMidlet`
- `runNams`

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*.

Note – The `runNams` application uses native application management software (AMS) APIs to run a MIDlet from an installed MIDlet suite and requires special building to be available.

For more information on how to build the phoneME Feature software to use native AMS and the `runNams` command, see the *Sun Java Wireless Client Build Guide*.

▼ Using the `runMidlet` Command

Several arguments can be used with the `runMidlet` command, but this *Getting Started Guide* provides only two, as shown below. 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 following arguments:

   - *suiteNumber* | *suiteID*

     The number or storage identifier given to the MIDlet suite when it was installed.

Note – Suite numbers and storage identifiers are displayed by the `listMidlet` command.

▼ 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 following arguments:

   - *suiteNumber* | *suiteID*

     The number or storage identifier given to the MIDlet suite when it was installed.

     Note – Suite numbers and storage identifiers are displayed by the `listMidlet` command.

   - `all`

     Requests that all MIDlet suites be removed.

Index

A

all
  PCSL build target, 14
  phoneME Feature software build target, 28

B

builds
  PCSL, 16

C

CLDC Hotspot Implementation, running, 25
configuration options
  PCSL
    FILE_MODULE, 15
    MEMORY_MODULE, 15
    NETWORK_MODULE, 15
    PCSL_CHUNKMEM_DIR, 15
    PCSL_CHUNKMEM_IMPL, 15
    PRINT_MODULE, 15
    USE_DEBUG, 15
  phoneME Feature software
    SUBSYSTEM_LCDUI_MODULES, 29
    USE_CLDC_11, 29
    USE_DEBUG, 30
    USE_FIXED, 29
    USE_I3_TEST, 30
    USE_JAVA_DEBUGGER, 30
    USE_JAVA_PROFILER, 30
    USE_MULTIPLE_ISOLATES, 29
    USE_SERVER_SOCKET, 30
    USE_SSL, 29

D

debug build mode, 22
doc
  PCSL build target, 14
docs_html phoneME Feature software build target, 28

E

environment variables
  See variables

F

FILE_MODULE
  PCSL configuration option, 15

I

installMidlet script, 34

J

JDK_DIR, phoneME Feature software environment variable, 8, 10, 21

M

MEMORY_MODULE PCSL configuration option, 15
MIDlet suites
  running, 36
MIDlets
  OTA installer, 34

N

NETWORK_MODULE PCSL configuration option, 15

O

OTA installer MIDlet, 34

P

PCSL_CHUNKMEM_DIR PCSL configuration option, 15

PCSL_CHUNKMEM_IMPL PCSL configuration option, 15

phoneME Feature software
build targets, 28

PRINT_MODULE PCSL configuration option, 15

R

running CLDC Hotspot Implementation, 25

running MIDlet suites, 36

S

scripts
installMidlet, 34

SUBSYSTEM_LCDUI_MODULES phoneME Feature software configuration option, 29

T

targets
PCSL build targets
all, 14
doc, 14
phoneME Feature software
all, 28
docs_html, 28

U

USE_CLDC_11 phoneME Feature software configuration option, 29

USE_DEBUG PCSL configuration option, 15

USE_DEBUG phoneME Feature software configuration option, 30

USE_FIXED phoneME Feature software configuration option, 29

USE_I3_TEST phoneME Feature software configuration option, 30

USE_JAVA_DEBUGGER phoneME Feature software configuration option, 30

USE_JAVA_PROFILER phoneME Feature software configuration option, 30

USE_MULTIPLE_ISOLATES phoneME Feature software configuration option, 29

USE_SERVER_SOCKET phoneME Feature software configuration option, 30

USE_SSL phoneME Feature software configuration option, 29

V

Variables
Setting on a Windows Platform, 9

variables
phoneME Feature software environment
JDK_DIR, 8, 10, 21