



Tools Guide

Sun Java™ Wireless Client Software, Version 2.1
Java Platform, Micro Edition

Copyright © 2008 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

THIS PRODUCT CONTAINS CONFIDENTIAL INFORMATION AND TRADE SECRETS OF SUN MICROSYSTEMS, INC. USE, DISCLOSURE OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SUN MICROSYSTEMS, INC.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties.

Sun, Sun Microsystems, the Sun logo, Java, Solaris, HotSpot, J2ME, J2SE, J2EE, Java Developer Connection, Java Community Process, JCP, Javadoc, JDK, JavaCall, Java Card, phoneME and the Java Coffee Cup logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

OpenGL is a registered trademark of Silicon Graphics, Inc.

The PostScript logo is a registered trademark of Adobe Systems, Incorporated.

Products covered by and information contained in this service manual are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2008 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, États-Unis. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuelle relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plusieurs des brevets américains listés à l'adresse <http://www.sun.com/patents> et un ou plusieurs brevets supplémentaires ou les applications de brevet en attente aux États-Unis et dans d'autres pays.

CE PRODUIT CONTIENT DES INFORMATIONS CONFIDENTIELLES ET DES SECRETS COMMERCIAUX DE SUN MICROSYSTEMS, INC. SON UTILISATION, SA DIVULGATION ET SA REPRODUCTION SONT INTERDITES SANS L'AUTORISATION EXPRESSE, ECRITE ET PREALABLE DE SUN MICROSYSTEMS, INC.

Droits du gouvernement des États-Unis - logiciel commercial. Les droits des utilisateur du gouvernement des États-Unis sont soumis aux termes de la licence standard Sun Microsystems et aux conditions appliquées de la FAR et de ces compléments.

Cette distribution peut inclure des éléments développés par des tiers.

Sun, Sun Microsystems, le logo Sun, Java, Solaris, HotSpot, J2ME, J2SE, J2EE, Java Developer Connection, Java Community Process, JCP, Javadoc, JDK, JavaCall, Java Card, phoneME et le logo Java Coffee Cup sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux États-Unis et dans d'autres pays.

UNIX est une marque déposée aux États-Unis et dans d'autres pays et sous licence exclusive de X/Open Company, Ltd.

Intel est une marque déposée de Intel Corporation ou de sa filiale aux États-Unis et dans d'autres pays.

OpenGL est une marque déposée de Silicon Graphics, Inc.

Le logo PostScript est une marque de fabrique ou une marque déposée de Adobe Systems, Incorporated.

Les produits qui font l'objet de ce manuel d'entretien et les informations qu'il contient sont régis par la législation américaine en matière de contrôlé des exportations et peuvent être soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes biologiques et chimiques ou du nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers des pays sous embargo des États-Unis, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régi par la législation américaine en matière de contrôlé des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DÉCLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISÉE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITÉ MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIÈRE OU A L'ABSENCE DE CONTREFAÇON.

Contents

Preface	vii
1. Managing Public Keys of Certificate Authorities	1
Running MEKeyTool	2
▼ Using the MEKeyTool Utility	2
ME Keystores	2
Creating and Managing Multiple ME Keystores	3
Creating Alternate ME Keystores	3
Managing Alternate ME Keystores	3
▼ Using Alternate Keystores to Run Executables	4
Importing a Key	4
Listing Available Keys	5
Deleting a Key	6
Replacing a Key	7
MEKeyTool Summary	7
MEKeyTool Options	8
2. Signing a MIDlet Suite's JAR File	9
Instructions for Using JadTool	9
▼ Using the JadTool Utility	10

Handling Expired Certificates	12
JadTool Summary	12
JadTool Utility Options	13
3. Converting Images to Raw Format	15
4. ROMizing Skin Definitions	17
SkinRomizationTool	18
run_skinromizer Build Target	19
Generating the skin.bin File	19
5. Running the Java Wireless Client Software	21
Using the OTA Installer	21
Running an Installed MIDlet	22
Using runMidlet on a Windows/i386 Platform	23
▼ To launch runMidlet	23
Using the Native AMS	24
Listing Installed MIDlet Suites	25
Removing an Installed MIDlet Suite	25
Using the Java Programming Language AMS	26
Listing an Installed MIDlet Suite	26
Removing an Installed MIDlet Suite	27
Emulating the Graphical AMS of a MIDP Phone	27
6. Debugging a MIDlet	29
The Debugging Process	29
7. Automatically Testing MIDlet Suite Management and Execution	31
autotest Script	31
autotestm Script	32

8. Running i3 Tests	33
A. Upgrading Configuration Files	35
Upgrade Localization Strings Tool	35
UpgradeROMizedProperties Tool	36
Glossary	37
Index	41

Preface

The *Tools Guide* describes how to use the Sun Java™ Wireless Client software tools and other executables to configure, run, and test an implementation of the Java Wireless Client software. These executables are not intended for end users. They are for platform developers, system developers, and systems integrators. The executables are unsupported.

Because these unsupported executables are for developers, not end users, they use simple option parsing. The simple parsing of executables might lead to reported error conditions that are not user friendly. The most confusing error condition occurs because the executables only ensure that option values are formatted according to the specification of the underlying MIDP APIs. They accept any potentially valid value for an option, even if the value looks like another option or a value for another option. Like the `java` command from the Java Platform, Standard Edition (Java SE platform), they do not reject any positive case, which they would have to do to make error messages more user friendly.

Note – Sun Microsystems has simplified the naming schemes for the various Java platforms. Java Platform, Enterprise Edition (Java EE) was formerly Java 2 Platform, Enterprise Edition (J2EE™). Java Platform, Standard Edition (Java SE) was formerly Java 2 Platform, Standard Edition (J2SE™). Java Platform, Micro Edition (Java ME) was formerly Java 2 Platform, Micro Edition (J2ME™).

References in this guide to specific documents, specifications, and products that were released when the old naming scheme was in use retain their original names. General references to Java platforms in this guide use the new, simplified naming scheme.

Before You Read This Guide

Readers using this guide must be familiar with the *MIDP 2.1 Specification*. The specification is available from <http://www.jcp.org/>.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials available through such sites.

This guide also assumes that the Java Wireless Client software is installed and an implementation reference port is built as described in the *Build Guide*.

How This Guide Is Organized

This book contains the following chapters:

Chapter 1 describes how to manage the certificate authority keys needed for secure network protocols and MIDlet authorization.

Chapter 2 describes how to sign a MIDlet suite's Java Archive (JAR) file.

Chapter 3 contains information about converting images to a raw format to optimize performance.

Chapter 4 covers the Skin ROMizer tool which is used during the build.

Chapter 5 describes how to use the command-line executables to manage and run MIDlet suites.

Chapter 6 describes how to connect Java Wireless Client to a debugger.

Chapter 7 describes how to automatically test MIDlet suite management and execution.

Chapter 8 describes how to use a MIDlet that lists and runs certain tests, called I3 tests.

Appendix A describes tools for upgrading configuration files from previous versions of Java Wireless Client.

Related Documentation

The following documentation is included with this release of the Java Wireless Client software:

Application	Title
All	<i>Release Notes</i>
Porting	<i>Porting User's Guide</i>
Building	<i>Build Guide</i>
Configuration and Testing Tools	<i>Tools Guide</i>
Multitasking Integration and Policies	<i>Multitasking Guide</i>
Customizing Adaptive User Interface Technology (skins)	<i>Skin Author's Guide to Adaptive User Interface Technology</i>
Viewing reference documentation created by the Javadoc™ tool	<i>Java API Reference</i>
Viewing reference documentation created by the Doxygen tool	<i>Native API Reference</i>

Typographic Conventions Used in This Guide

Typeface	Meaning	Examples
Courier AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. % You have mail.
Bold AaBbCc123	What you type, when contrasted with on-screen computer output	% su Password:
	Important parts of a code sample	
<i>Italic</i> AaBbCc123	Book titles, new words or terms, words to be emphasized	Read Chapter 6 in the <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be super user to do this.
	Command-line variable; replace with a real name or value	To delete a file, type <code>rm filename</code> .

Accessing Sun Documentation Online

The Java Developer Connection™ program web site enables you to access Java platform technical documentation at <http://java.sun.com/>.

Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. Provide feedback to Sun at <http://java.sun.com/docs/forms/sendusmail.html>.

Managing Public Keys of Certificate Authorities

Java Wireless Client uses public keys from a Certificate Authority (CA) to check the validity of web sites and signed MIDlet suites. When Java Wireless Client uses a secure protocol to access a web site, the site provides a certificate typically signed by a CA. Similarly, signed MIDlet suites contain a certificate that is typically signed by a CA. By signing a certificate, a CA vouches for the web site or MIDlet suite.

Java Wireless Client checks the validity of a certificate by using the CA's public key. You manage the CA public keys for your Java Wireless Client port with the `MEKeyTool` utility. The utility is functionally similar to the `keytool` utility provided with the Java SE platform.

This chapter describes how to use `MEKeyTool`. This chapter contains the following sections:

- [Running MEKeyTool](#)
- [ME Keystores](#)
- [Importing a Key](#)
- [Listing Available Keys](#)
- [Deleting a Key](#)
- [Replacing a Key](#)
- [MEKeyTool Summary](#)

In the following sections, *installDir* represents the Java Wireless Client installation directory.

Running MEKeyTool

MEKeyTool is packaged in a JAR file, `MEKeyTool.jar`, in the `installDir/midp/bin` directory.

▼ Using the MEKeyTool Utility

1. Open a command prompt or terminal window.

2. Change your current directory to `installDir/midp`.

For example, if Java Wireless Client software is installed in the directory `$HOME/jwc2.1`:

```
$ cd $HOME/jwc2.1/midp
```

3. Run **MEKeyTool** with the `java` command, the `-jar` option, and any options to **MEKeyTool**.

The commands and options to the MEKeyTool utility are described in [“MEKeyTool Summary” on page 7](#). Using no options provides help. For example, if the `java` command is on your `PATH`, use the following command to display help:

```
$ java -jar bin/MEKeyTool.jar
```

ME Keystores

The MEKeyTool utility keeps the CA public keys in an *ME keystore*, a file that holds the keys in a format that the Java ME platform can use. When using a secure protocol, the Java Wireless Client software implementation expects the ME keystore to be the file `installDir/midp/appdb/_main.ks`. Java Wireless Client software includes the file and the key of one CA. By default the MEKeytool utility uses this file when you add, delete, or list keys.

An ME keystore has a different format from the Java SE platform’s J2EE Connector Architecture (Connector Architecture) keystore file. You cannot use the MEKeytool utility on a Connector Architecture keystore. For example, if you try to use the MEKeytool utility to view public keys in a Connector Architecture keystore, the utility displays an error message that the keystore is corrupted. The message indicates that the keystore is not in a format that the MEKeyTool utility can read. However, the keystore is in the correct format for the Java SE platform tools.

Creating and Managing Multiple ME Keystores

In addition to managing the CA public keys in the default ME keystore, you can use the `MEKeyTool` utility to manage additional ME keystores. For example, during testing you might want to have multiple keystores to run against: One keystore could contain all the keys, a second keystore could contain a subset of keys, and a third keystore could contain an expired key.

Creating Alternate ME Keystores

Importing a key is one way to create a new ME keystore. If you import a key into a keystore that does not exist, the `MEKeyTool` utility creates it. See [“Importing a Key” on page 4](#) for instructions on how to import a key. A second way to create an ME keystore is to copy the keystore provided with Java Wireless Client software.

Managing Alternate ME Keystores

To manage a keystore other than the default, use the `-MEkeystore` option:

```
java -jar bin/MEKeyTool.jar -MEkeystore keystoreName ...
```

For example, if you created an ME keystore containing the keys needed to run a particular set of tests, `$HOME/myKeys/set2_test_keys.ks`, use the `MEKeyTool` command to manage that keystore:

```
java -jar bin/MEKeyTool.jar  
-MEkeystore $HOME/myKeys/set2_test_keys.ks ...
```

For most `MEKeyTool` commands, you receive an error message if the file that you provide as an argument to `-MEkeystore` does not exist. The only exception is importing a key, when `MEKeyTool` creates a new ME keystore with the name that you supply.

▼ Using Alternate Keystores to Run Executables

The Java Wireless Client software executables only use the ME keystore:

```
installDir/midp/appdb/_main.ks
```

To use an alternate keystore, you must perform the following steps *before* running an executable:

1. **Create a backup copy of the existing *installDir/midp/appdb/_main.ks*. If you already have a copy of this file, go to the next step.**

For example, if your current directory is *installDir*, run the following command:

```
$ cp midp/appdb/_main.ks myKeys/orig_main.ks
```

2. **Copy your alternate keystore to *installDir/midp/appdb/_main.ks*.**

For example, to use the keys in the file `$HOME/myKeys/set2_test_keys.ks`, for this run of the Java Wireless Client software executable, run the following command:

```
$ cp myKeys/set2_test_keys.ks midp/appdb/_main.ks
```

Importing a Key

The ME keystore comes with few keys. If you cannot visit some web sites or install some MIDlets, you can add the required keys to the keystore. You must also add new keys to the keystore when existing keys expire. First delete the expired key, then add the new one.

Add a key to an ME keystore by importing it from the Connector Architecture keystore that comes with the Java SE platform or from a keystore that you create. For more information on the keystore that comes with the Java SE platform, see <http://java.sun.com/j2se/1.4/docs/tooldocs/win32/keytool.html>.

The default Connector Architecture keystore for the `MEKeyTool` utility is `$HOME/.keystore`.

Note – If you use a Connector Architecture keystore other than the default, the new keystore might require a password.

When you add a key to an ME keystore, you can associate a security domain with it. The Java Wireless Client software can assign that domain to any MIDlet suite for which the owner of the public key was a signer. If you do not provide a domain, the public key is associated with the domain called `identified`.

The option that imports a key is `-import`. For example, to add a key with an alias `dummyca` from the Connector Architecture keystore

`../bin/j2se_test_keystore.bin` to the ME keystore

`$HOME/myKeys/set2_test_keys.ks`:

```
$ java -jar bin/MEKeyTool.jar -import -alias dummyca
-keystore ../bin/j2se_test_keystore.bin
-MEkeystore myKeys/set2_test_keys.ks
```

If the Connector Architecture keystore has a password, `keystorepwd`, and you want to assign the public key to the operator domain, use the command:

```
$ java -jar bin/MEKeyTool.jar -import -alias dummyca
-keystore ../bin/j2se_test_keystore.bin -storepass keystorepwd
-MEkeystore myKeys/set2_test_keys.ks -domain operator
```

Listing Available Keys

An ME keystore organizes the keys that it contains by giving each one a number. For each key, the keystore also holds the name of the entity to whom the public key belongs, the time over which the key is valid, and the domain associated with the key. The `MEKeyTool -list` command displays information for each key in a particular keystore.

The following example lists the contents of the ME keystore

`$HOME/myKeys/set1_test_keys.ks`:

```
$ java -jar bin/MEKeyTool.jar -list
-MEkeystore myKeys/set1_test_keys.ks
```

Key 1

```
Owner: C=US;O=RSA Data Security, Inc.;OU=Secure Server
Certification Authority
```

```
Valid from Tue Nov 08 16:00:00 PST 1994 to Thu Jan 07 15:59:59 PST
2010
```

```
Security Domain: identified
```

Key 2

```
Owner: O=Sun Microsystems;C=myserver
```

```
Valid from Sat Aug 03 00:43:51 PDT 2002 to Tue Jul 31 00:43:51 PDT
2012
```

```
Security Domain: operator
```

Deleting a Key

When keys expire, you must delete them from the keystore and add their replacements. You can also delete unused keys. For example, if you added the public key of a test site with a self-signed certificate during testing, you can delete that key when testing is completed.

The `-delete` command to the `MEKeyTool` utility removes a key from an ME keystore. The `-delete` command requires one of the following options:

- **-owner** *ownerName*

Sets the string that describes the owner of the public key in a given keystore. Use the `-list` command to print information about each key in the keystore. The string in the command must match the one printed when you use the `-list` command to the `MEKeyTool` utility. See [“Listing Available Keys” on page 5](#) for more information.

- **-number** *keyNumber*

Sets the number that a given keystore has assigned to each of its keys. The number is greater than or equal to one. Use the `-list` command to print the number that the keystore has assigned to each of its keys. See [“Listing Available Keys” on page 5](#) for more information.

The following examples show two ways to delete a key from the ME keystore `$HOME/myKeys/set1_test_keys.ks` (the keystore used in [“Listing Available Keys” on page 5](#)):

- Deleting a key by using its key number-

```
$ java -jar bin/MEKeyTool.jar -delete -number 1
-MEkeystore myKeys/set1_test_keys.ks
```

- Deleting a key by using its owner name-

```
$ java -jar bin/MEKeyTool.jar -delete -owner "OU=Secure Server
Certification Authority;O=RSA Data Security, Inc.;C=US"
-MEkeystore myKeys/set1_test_keys.ks
```

Replacing a Key

Some situations require that you replace a key (such as when a key expires). To replace a key, first delete the old key, then import the new key.

Note – If you import the new key before deleting the old one, the `MEKeyTool` utility displays an error message that the owner of the key already has a key in the ME keystore.

See “Deleting a Key” on page 6 for instructions on how to delete a key.

See “Importing a Key” on page 4 for instructions on how to import the new key.

MEKeyTool Summary

The following summarizes the `MEKeyTool` utility command and options:

```
java -jar MEKeyTool.jar

  -delete [ -MEkeystore MEKeystore ]
( -owner ownerName | -number keyNumber )

  [-help]

  -import [ -MEkeystore MEKeystore ] -alias keyAlias
[ -keystore CAKeystore ] [ -storepass storePassword ]
[ -domain domain ]

  -list [ -MEkeystore MEKeystore ]
```

Note – In the command line above, the variable `CAKeystore` stands for “Connector Architecture keystore” and not “Certificate Authority keystore.”

MEKeyTool Options

The MEKeyTool utility supports the following options:

- **none**

Runs the tool without options returns the same information as the `-help` option.

- **-delete** [**-MEkeystore** *MEKeystore*]
(**-owner** *ownerName* | **-number** *keyNumber*)

Deletes the key with *ownerName* or *keyNumber* from *MEKeystore*. If *MEKeystore* is not provided, its default, *installDir/midp/appdb/_main.ks*, is used.

You can provide either *ownerName* or *keyNumber*, but not both. You can find the valid values for them by running the MEKeyTool utility with the `-list` command.

- **-help**

Prints a usage summary.

- **-import** [**-MEkeystore** *MEKeystore*] **-alias** *keyAlias* [**-domain** *domain*]
[**-keystore** *CAKeystore*] [**-storepass** *storePassword*]

Imports a public key from *CAKeystore* into *MEKeystore*, and associates the public key with *domain*. If *CAKeystore* is not provided, its default, *\$HOME/.keystore*, is used (where *\$HOME* is the user's home directory). If *MEKeystore* is not provided, its default, *installDir/midp/appdb/_main.ks*, is used. If *domain* is not provided, its default domain, called *identified*, is used.

If *CAKeystore* requires a password, you must provide *storePassword*.

Note – In the command line above, the variable *CAKeystore* stands for “Connector Architecture keystore” and not “Certificate Authority keystore.”

- **-list** [**-MEkeystore** *MEKeystore*]

Lists the number, owner, and validity period, and domain of each key in *MEKeystore*. If *MEKeystore* is not provided, the default, *installDir/midp/appdb/_main.ks*, is used.

Signing a MIDlet Suite's JAR File

Trusted MIDlets can typically access more protected functionality than untrusted ones. As a result, establishing trust is important for MIDlet suites that use security-sensitive APIs. Signing a MIDlet suite's JAR file allows the suite to be trusted. A JAR file is signed with the `JadTool` utility provided with Java Wireless Client software.

The `JadTool` utility signs a JAR file by adding a certificate and the JAR file's digital signature to a Java Application Descriptor (JAD) file. Adding a certificate and a JAR file's digital signature to a JAD file are separate steps. You must complete both steps to sign a JAR file. The steps are in ["Instructions for Using JadTool"](#) on page 9.

You can also use the `JadTool` utility to obtain information about a certificate in a JAD file. The information can include the name of the entity that issued the certificate, the certificate's serial number, the dates between which is it valid, and its MD5 and SHA fingerprints.

This chapter describes the use of the `JadTool` utility, including an example. It has the sections:

- [Instructions for Using JadTool](#)
- [JadTool Summary](#)

Instructions for Using JadTool

This section explains how to use the `JadTool` utility through an example that signs a hypothetical MIDlet suite, `ImaginaryMIDlet`.

Note – `ImaginaryMIDlet` is not an actual MIDlet suite. No `ImaginaryMIDlet` files are included with this release.

The example uses the key pair provided with Java Wireless Client software. The key pair is in the `j2se_test_keystore.bin` file, which is a keystore managed with the Java SE platform's `keytool` utility. For information on the `keytool` utility, see <http://java.sun.com/j2se/1.4/docs/tooldocs/win32/keytool.html>.

After you build an implementation of the Java Wireless Client software, `j2se_test_keystore.bin` is located in the `installDir/midp/bin` directory. The password for the file is `keystorepwd`. The alias of the key pair is `dummyca`. The private key password is `keypwd`. The file is provided for testing purposes.

For MIDlet suites on end-user devices, use an RSA key pair backed by a certificate or certificate chain from a certificate authority. You must import the certificate or certificate chain into a Java SE platform's keystore with the Java SE platform's `keytool` utility.

The `JadTool` utility is packaged in a JAR file, `JadTool.jar`, in the `installDir/midp/bin` directory.

▼ Using the `JadTool` Utility

1. **Open a command prompt or terminal window.**
2. **Change your current directory to the directory that holds your MIDlet's JAR and JAD files.**

For example, if the JAR and JAD files are in the directory `$HOME/myMIDlets/`, issue the following command:

```
$ cd $HOME/myMIDlets/
```

3. **Add the certificate for your public key pair to the JAD file using the `JadTool` utility.**

The `JadTool` utility adds the certificate as the value of an attribute named `MIDlet-Certificate-m-n`, where *m* is the number of the certificate chain (it defaults to one but you can provide a different number with the `-chainnum` switch), and *n* is an integer that, for new certificates, begins at one and increments by one each time you add a new certificate to the JAD file.

For example, if `installDir` is `$HOME/jwc2.0`, the following command adds the certificate as the value of the attribute `MIDlet-Certificate-1-1` to the example JAD file:

```
$ java -jar $HOME/jwc2.0/midp/bin/JadTool.jar -addcert
-alias dummyca -storepass keystorepwd
-keystore $HOME/jwc2.0/midp/bin/j2se_test_keystore.bin
-inputjad ImaginaryMIDlet.jad -outputjad ImaginaryMIDlet.jad
```

4. (Optional) Verify that the certificate is added to the JAD file by using the JadTool utility to list the certificate in the JAD file.

```
$ java -jar $HOME/jwc2.0/midp/bin/JadTool.jar -showcert  
-certnum 1 -inputjad ImaginaryMIDlet.jad
```

```
Subject: C=US, ST=CA, L=Santa Clara, O=dummy CA, OU=JCT, CN=thehost
```

```
Issuer : C=US, ST=CA, L=Santa Clara, O=dummy CA, OU=JCT, CN=thehost
```

```
Serial number: 3d3ece8a
```

```
Valid from Wed Jul 24 08:58:02 PDT 2002 to Sat Jul 21 08:58:02 PDT 2012
```

```
Certificate fingerprints:
```

```
MD5: 87:7f:5e:64:c8:dd:b4:bf:35:39:76:87:99:9b:68:82
```

```
SHA: 9d:c0:88:ce:08:83:cd:e6:fe:13:8b:26:f6:b4:df:e2:da:3c:25:98
```

5. If you have a key pair backed by a certificate chain, import the intermediate certificates.

Import the intermediate certificates using the `JadTool` utility with the `-addcert` switch shown in [Step 3](#), taking care to use the correct chain order.

For example:

The XXXX company provides a certificate that vouches for your key pair, the WidgetCertificates company vouches for the XXXX certificate, and VeriSign vouches for the WidgetCertificates certificate.

Import the XXXX certificate followed by the WidgetCertificate. The XXXX certificate is `MIDlet-Certificate-1-2` and the WidgetCertificate certificate is `MIDlet-Certificate-1-3`.

Note – You do not import the certificate of the root CA. In this example, the certificate is from VeriSign. The root certificate is on the device.

6. Sign the JAR file using the JadTool utility.

The `JadTool` utility signs the JAR file, base64 encodes the signature, and stores it as the value of the `MIDlet-Jar-RSA-SHA1` attribute of the output JAD file.

Note – The key used to sign the JAR file must be from the same Connector Architecture keystore entry as key pair specified in [Step 3](#). The `JadTool` utility does not check that the JAR file is signed with a keystore entry that has a certificate in the JAD file.

For example:

```
$ java -jar $HOME/jwc2.0/midp/bin/JadTool.jar -addjarsig
-keystore $HOME/jwc2.0/midp/bin/j2se_test_keystore.bin
-alias dummyca -storepass keystorepwd -keypass keypwd
-jarfile ImaginaryMIDlet.jar -inputjad ImaginaryMIDlet.jad
-outputjad ImaginaryMIDlet.jad
```

Handling Expired Certificates

A JAD file can have more than one certificate, but it can hold the signature for only one JAR file. When a certificate in the JAD file expires, you must add a new certificate and re-sign the JAR file. When re-signing the JAR file, the `JadTool` utility overwrites the current digital signature with the new one.

JadTool Summary

The following summarizes the `JadTool` utility command and options:

```
java -jar JadTool.jar

[ -addcert -alias keyAlias [ -keystore keystore ] [ -storepass password ]
  [ -certnum certNumber ] [ -chainnum chainNumber ]
  [ -encoding encoding ] -inputjad inputJadFile -outputjad outputJadFile ]

[ -addjarsig [ -jarfile jarFile ] -alias keyAlias [ -keystore keystore ]
  -storepass password -keypass keyPassword
  [ -encoding encoding ] -inputjad inputJadFile -outputjad outputJadFile ]

[ -help ]

[ -showcert
  [ ( [ -certnum certNumber ] [ -chainnum chainNumber ] ) | -all ]
  [ -encoding encoding ] -inputjad inputJadFile ]
```

JadTool Utility Options

The `JadTool` utility supports the following options:

- **none**

Running the tool without options returns the same information as the `-help` option.

- **-addcert -alias *keyAlias* [-keystore *keystore*] [-storepass *password*] [-chainnum *chainNumber*] [-certnum *certNumber*] [-encoding *encoding*] -inputjad *inputJadFile* -outputjad *outputJadFile***

Adds a certificate to a JAD file. To do this, this utility first creates the certificate from the entry identified by *keyAlias* in *keystore*. The *keystore*, if provided, must be a Connector Architecture keystore (a file containing data such as key entries in a format that the Java SE platform can use). If *keystore* is not provided, its default, `$HOME/.keystore`, is used. If *keystore* requires a password to access its contents, *password* must be provided.

After creating the certificate and attribute name, this utility concatenates the contents of *inputJadFile* with the new certificate and writes it as *outputJadFile*.

Note – You can use the same file for the *inputJadFile* and *outputJadFile*.

The certificate is in the JAD file as the value of an attribute named `MIDlet-Certificate-m-n`, where:

- *m* is *chainNumber*, or 1 if it is not provided. A JAD file can contain more than one certificate chain.
- *n* is *certNumber*. The value *certNumber* depends on whether the new certificate replaces an existing certificate. If the certificate is a replacement, then *certNumber* must be the number of the certificate to replace. For example, if the new certificate would replace the one stored as the value of attribute `MIDlet-Certificate-1-3`, then *certNumber* must be 3. If the certificate is new, *certNumber* is ignored.

If *inputJadFile* uses an encoding other than UTF-8 (ASCII with unicode escapes), *encoding* must be specified. This utility uses the same encoding for reading *inputJadFile* and writing *outputJadFile*.

- **-addjarsig [-jarfile *jarFile*] -alias *keyAlias* [-keystore *keystore*] -storepass *storePassword* -keypass *keyPassword* [-encoding *encoding*] -inputjad *inputJadFile* -outputjad *outputJadFile***

Creates a digital signature for *jarFile*. If *jarFile* is not specified, the value of the `MIDlet-Jar-URL` attribute from *inputJadFile* is used. The attribute's value must be a valid HTTP URL.

This utility creates a digital signature for the JAR file using the private key identified by *keyAlias* in *keystore*. If *keystore* is not provided, its default is `$HOME/.keystore`. This utility gets the key from *keystore* using *storePassword* and *keyPassword*, and creates the signature with it using the EMSA-PKCS1-v1_5 encoding method of PKCS #1, version 2.0. See RFC 2437 at <http://www.ietf.org/rfc/rfc2437.txt>.

After creating the signature, this utility concatenates the contents of *inputJadFile* with the signature, and writes it as *outputJadFile*. The signature is base64 encoded, and is in the output JAD file as the value of the `MIDlet-Jar-RSA-SHA1` attribute.

If *inputJadFile* uses an encoding other than UTF-8 (ASCII with unicode escapes), *encoding* must be specified. This utility uses the same encoding for reading *inputJadFile* and writing *outputJadFile*.

■ **-help**

Prints a usage summary.

■ **-showcert** [([**-certnum** *certNumber*] [**-chainnum** *chainNumber*]) | **-all**] [**-encoding** *encoding*] **-inputjad** *inputJadFile*

Prints information about either all certificates, or the certificate that corresponds to the given *certNumber* and *chainNumber* in the *inputJadFile*. The option `-all` cannot be combined with the `-certnum` and `-chainnum` options.

The *chainNumber* of a certificate is the *m* in the JAD file's `MIDlet-Certificate-m-n` attribute, while the *certNumber* is the *n*. For example, to show the certificate that is the value of attribute `MIDlet-Certificate-2-3`, the *chainNumber* must be 2 and *certNumber* must be 3. If *certNumber* or *chainNumber* are not provided (and the `-all` option is not used), the utility uses a 1.

The information printed includes the certificate's subject, issuer, serial number, dates between which it is valid, and fingerprints (md5 and SHA). The attributes in the subject and issuer names are shown in reverse order from what is in the certificate (a side effect of using the Java SE platform certificate API). As a result, the names might not match what is returned from other tools that display a certificate's subject and issuer names.

If *inputJadFile* uses an encoding other than UTF-8 (ASCII with unicode escapes), *encoding* must be specified. The tool uses the same encoding for reading *inputJadFile* and writing *outputJadFile*.

Converting Images to Raw Format

To make the Java AMS run as fast as possible, the Java Wireless Client software enables images to be loaded from raw data files instead of decoded from PNG files. This option is turned on at build time by specifying `USE_RAW_AMS_IMAGES=true`.

The `ImageToRawTool` converts PNG images to an appropriate raw format. The exact raw format depends on the target build platform.

The source code for `ImageToRawTool` is in:

InstallDir/midp/src/tool/imageutil/classes/com/sun/midp/imageutil.

A JAR file containing the tool is created and used at build time. It is located in the output binaries folder directory:

InstallDir/output/midp/bin/platform

`output` is the top-level directory designated as the location for your Java Wireless Client software build output and *platform* is the target platform. For example, the on Linux the target platform could be `i386` or `arm`; on Win32, it is the JavaCall™ API.

Note – The *InstallDir/output* directory is named by convention. You can name the output directory for your Java Wireless Client software build anything you choose.

The `ImageToRawTool` options are as follows:

```
[ -help]
[ -debug]
-format formatXMLFile
-out outputImagesDirectory
imagesToConvert
```

The raw image format is described in *formatXMLFile*, according to *configuration.dtd*, which works in a similar way to *skin.xml*.

For example, the *linux_fb* platform uses the following raw image format:

```
<rawimage
  Format="Putpixel"
  Colors="565"
  Endian="Little"
/>
```

The descriptions of raw image formats for each platform supported in the Java Wireless Client software are provided in the *rawimage.xml* files located in *InstallDir/midp/src/configuration/configuration_xml/platform*.

In the above path, *platform* could be *win32*, *arm*, the *javacall* porting layer, *linux_fb*, or *linux_qte*.

ROMizing Skin Definitions

The Sun Java Wireless Client software provides AUIT (Chameleon) skin designers with the ability to make changes to an existing skin and then generate a new one, without needing to rebuild the full software product. This is done through the use of the `make build` target, `run_skinromizer`, and the resulting binary output file, `skin.bin`.

The process of generating an initial skin and then updating it occurs in several steps, some of which are automated by the Sun Java Wireless Client build process. To build a skin and then modify, follow these general steps:

1. The skin designer creates graphics for the skin and defines skin property settings in the `skin.xml` file. For more information, see the *Sun Java Wireless Client Skin Author's Guide to Adaptive User Interface Technology*.
2. The Sun Java Wireless Client software product is built. During this process, the `skin.xml` file is merged with other XML configuration files and processed by the Configurator tool. For more information on the Configurator Tool, see the *Sun Java Wireless Client Porting Guide*.
3. The `SkinRomizationTool` operates on the output generated by the Configurator Tool, to romize graphic images and create a binary output file of the `skin.xml` properties file, called `skin.bin`. The `skin.bin` file is written to the directory `InstallDir/output/midp/lib` (where `output` is the top-level directory you designate for the output of your build.)

Note – Images defined as “ROMized” in the `skin.xml` file are romized into C files, and the Sun Java Wireless Client software source files are built into a finished Java language binary executable. The only change to the build process from previous versions is output of the `skin.bin` file, which improves the skinning process.

4. Once the `skin.bin` file is generated, it exists as a binary file that can be installed onto a device, where it works seamlessly with the existing Java Wireless Client software binary executable.

Before installing `skin.bin` onto a device, the skin designer is able to modify the properties in the `skin.xml` file and generate a new binary `skin.bin` file as many times as needed, without rebuilding the full Java Wireless Client software each time.

SkinRomizationTool

Although not normally run manually from the command line, the parameters of the `SkinRomizationTool` can be modified in the makefiles to be used during the build process.

InstallDir/midp/src/tool/chameleon/classes/com/sun/midp/skinromization/

During the build process, the `SkinRomizationTool` is built into the `SkinRomizationTool.jar` file.

The `SkinRomizationTool` command-line options are as follows:

- `-xml` *localXMLFile*
- `-imagedir` *skinImagesDirName*
- `-outjava` *localOutputJavaFile*
- `-outc` *localOutputCFile*
- `-debug`

localXMLFile is the name of the XML file to be processed by `SkinRomizationTool`. During the build, it is the name of the merged XML file produced by the Configurator.

skinImagesDirName is the directory containing skin images. `SkinRomizationTool` looks in this directory for any images it needs to ROMize.

localOutputJavaFile is the name of the output Java programming language file generated by `SkinRomizationTool`.

localOutputCFile is the name of the output C file generated by `SkinRomizationTool`.

The `-debug` option prints some debugging information during the XML file processing.

run_skinromizer Build Target

Once the `skin.bin` binary file is created, the skin designer is able to make changes to the skin and redisplay it to check those changes. This gives the skin designer increased flexibility by reducing the overhead needed to view changes made to the skin.

Changes made to the graphics files themselves are not picked up by regenerating the `skin.bin` file if your image files are set to `Romized=true` in your platform's `skin.xml` file. If your graphic images are defined in the `skin.xml` file as `Romized=false`, a skin designer can easily change the look of a gauge from one with round corners to one with square corners. For example, the skin designer can simply replace the image file called by the gauge property `GAUGE_IMAGE_BG` and then regenerate the `skin.bin` file.

However, if graphic images are set to `Romized=true`, regenerating the `skin.bin` file will not pick up the change made to those graphic images. The new graphic images are displayed in the skin only after the Sun Java Wireless Client software has been rebuilt.

For more information on changing properties and working with the `skin.xml` file, and how to work effectively with ROMized and non-ROMized graphic images, see the *Sun Java Wireless Client Skin Author's Guide to Adaptive User Interface Technology*.

Generating the `skin.bin` File

To generate the `skin.bin` file, run the `make` command on the command line, using `run_skinromizer` as a target. For example, on a Linux command line in the bash shell, enter the following command:

```
$ make run_skinromizer
```

This generates the `skin.bin` file and writes it to the following directory

```
InstallDir/output/midp/lib
```

`output` is the top-level directory you designate for the output of your software build.

Note – The `InstallDir/output` directory is named by convention. You can name the output directory for your Java Wireless Client software build anything you choose.

To properly generate a new `skin.bin` file, your build platform must be properly configured and set up. The `make run_skinromizer` command also requires the use of additional build variables.

For a complete explanation of how to use the `make run_skinromizer` command to generate a new skin, see the *Sun Java Wireless Client Skin Author's Guide to Adaptive User Interface Technology*.

Running the Java Wireless Client Software

Building an implementation of the Java Wireless Client software with the target `all` produces applications intended only for engineers who are porting Java Wireless Client to a new platform, or who are testing that port. The applications are unsupported. See the *Build Guide* for build instructions and the *Porting Guide* for porting instructions.

This chapter describes how to use the unsupported applications. It contains the following sections:

- [Using the OTA Installer](#)
- [Running an Installed MIDlet](#)
- [Using the Native AMS](#)
- [Listing Installed MIDlet Suites](#)
- [Removing an Installed MIDlet Suite](#)
- [Emulating the Graphical AMS of a MIDP Phone](#)

Using the OTA Installer

The graphical OTA installer is an installer used by end users. When an end user selects a link to a JAD or JAR file, the MIDlet-suite discovery application starts the installer. However, for developers and systems integrators, Java Wireless Client software provides a shell script, called `installMIDlet`, that runs the OTA Installer MIDlet from the command line and bypasses the discovery application. The script is located in the `installDir/midp/bin` directory.

The following summarizes the command line for the `installMidlet` script:

```
installMidlet url [urlLabel]
```

- *url*

The HTTP or HTTPS URL of a JAD or a JAR file. The MIDlet suite is installed from this URL.

- *urlLabel*

Optional label for the URL. The installer displays the label in the progress form.

The Java Wireless Client software OTA Installer only uses the HTTP and HTTPS protocols to install MIDlet suites.

Running an Installed MIDlet

The example code that runs a MIDlet from an installed MIDlet suite is `runMidlet`. The `runMidlet` application is located in `InstallDir/midp/bin` directory. The application is compiled from the `runMidlet.c` file in the `installDir/midp/src/ams/example/jams/native/targetPlatform` directory. Note that when you build the Java Wireless Client software with debugging, the application's name is `runMidlet_g`.

The following summarizes the command line for the `runMidlet` application:

```
runMidlet [vmArgs] [-debug] [-loop] [-classpathext path] suiteID  
  [MIDletClassName [arg0 [arg1 [arg2]]]]
```

The `runMidlet` application takes the following arguments:

- *vmArgs*

Arguments that are recognized by the virtual machine (VM), such as `-DSystemPropertyName`, `-int`, and `-comp`. See your Connected Limited Device Configuration (CLDC) documentation for more information.

- `-debug`

Starts the VM suspended in debug mode and waits for the debugger to connect. The internal system property `VmDebuggerPort` determines which port the VM uses to listen for debug commands.

- `-loop`

Runs the MIDlet in a loop until system shuts down.

- `-classpathext path`

Appends *path* to the classpath passed to the VM. The VM can then access classes from *path* as if they were ROMized.

- *suiteID*
A unique integer identifying the MIDlet suite and assigned to it when it was installed.
- *MIDletClassName*
The name of the MIDlet that the system runs in a loop until the system shuts down. If this argument is not provided and the suite has multiple MIDlets, `runMidlet` uses the first MIDlet from the suite.
- *arg0 arg1 arg2*
Arguments passed to the MIDlet.

Using `runMidlet` on a Windows/i386 Platform

Using `runMidlet` on a Windows/i386 platform differs from running on other platforms. Keep these following things in mind when using `runMidlet`:

- Start `runMidlet` from the directory `InstallDir\build_output\midp\bin\platform`.
`build_output` is the directory where you write the output from your Java Wireless Client software build process.
- To start the Application Manager, invoke `runMidlet` with no arguments. Do this instead of running `usertest`.
- You can also use `runMidlet` to install applications from URLs or files, to run tests, or perform other functions. For a brief description of the syntax and examples of `runMidlet`, use the help option:

```
$ runMidlet help
```
- Use the `tck` option `runMidlet` instead of running `autotest`.
- Use the `install` option of `runMidlet` instead of `installMidlet`.

▼ To launch `runMidlet`

1. **Change directory to** `InstallDir\build_output\midp\bin\platform`.
In the above, *platform* is your Windows target platform, for example, `i386`.
2. **Type** `runMidlet.exe manager`

Using the Native AMS

The example code that runs a MIDlet from an installed MIDlet suite using the Native AMS API is `runNams`. To build this application, Java Wireless Client must be built with `USE_NATIVE_AMS=true` and `USE_MULTIPLE_ISOLATES=true`.

The CLDC HI software must be built with `ENABLE_ISOLATES=true`. `runNams` is compiled from the `runNams.c` file in the `InstallDir/midp/src/ams/example/nams/native/share` directory. When you build the Java Wireless Client software with debugging, the application's name is `runNams_g`.

The four options of `runNams` have the following syntax:

- `runNams [vmArgs] -namsTestService`
 - `runNams [vmArgs] -runMainClass mainClass [args]`
 - `runNams [vmArgs] -runMidlet suiteID [MIDletClassName [args]]`
 - `runNams [vmArgs] -jamsTestMode suiteID [MIDletClassName [args]]`
- vmArgs* is the same as for `runMidlet`.

The options are as follows:

- `-namsTestService` runs the NAMS Test Service. To use this mode, you must change `MIDP_DIR/src/ams/nams/lib.gmk` in the following way:
`$(NAMS_DIR)/i3test/com/sun/midp/main/NamsTestService.java` must be included instead of a stubbed out version of this file. Please refer to the comments in `lib.gmk`. For all other modes, the stubbed out version of this file must be included.

In this mode, `runNams` will listen on a socket (port 13322) for incoming connections and execute the given commands (it can create and start, destroy, pause, resume and switch a midlet to foreground).

- `-runMainClass` runs an alternate main class.
- `-runMidlet` runs a MIDlet using the NAMS API "regular" mode, allowing you to run a MIDlet in a non-AMS isolate.
- `-jamsTestMode` runs a MIDlet in the AMS isolate using the JAMS mode API. This mode should only be used to run the `i3test` framework and `AutoTester`. For executing other MIDlets, use `-runMidlet`.

The *suiteID* argument is exactly the same as for `runMIDlet`.

Listing Installed MIDlet Suites

When you use a native application management system (AMS), the `listMidlets` example code lists the installed MIDlets. This example code is not available for the Java platform AMS.

The `listMidlets` application is located in the *InstallDir/midp/bin* directory. The `listMidlets` application is compiled from the `listMidlets.c` file in the *InstallDir/midp/src/ams/example/nams/native/targetPlatform* directory.

The `listMidlets` command takes no arguments:

```
$ listMidlets
```

When no MIDlet suites are installed, the example displays:

```
** No MIDlet Suites installed on phone
```

If MIDlet suites are installed, the listing displays the following information (the number in square brackets is the MIDlet suite number):

```
[1]
  Name: HelloMIDlet
  Vendor: HelloMIDlet
  Version: 1.0
  Description: (null)
  Security Domain: operator
  Verified: false
  Suite ID: 2
  JAD URL: http://host.sun.com/HelloMIDlet.jad
  JAR URL: http://host.sun.com/HelloMIDlet.jar
  Size: 2K
```

Removing an Installed MIDlet Suite

When you use a native AMS, use the `removeMidlet` application to remove an installed MIDlet suite. This example code is not available for the Java platform AMS.

The `removeMidlet` application is located in the *InstallDir/midp/bin* directory. The `removeMidlet` application is compiled from the `removeMidlet.c` file in the

InstallDir/midp/src/ams/example/nams/native/targetPlatform directory.

The following summarizes the command line for the `removeMidlet` application:

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

- *suiteID*

Unique integer given to the MIDlet suite when it was installed. Integers are displayed by the `listMidlet` example. See [“Using the Native AMS” on page 24](#) for information.

- `all`

Requests that all MIDlet suites be removed.

Using the Java Programming Language AMS

The following utilities for working with MIDlets are available only when building with the Java language AMS (that is, when setting the build flag `USE_NATIVE_AMS=false`):

- `listMidlets.sh`
- `removeMidlet.sh`

Note – To use these utilities under Win32, Cyg4me (or Cygwin) must be installed.

Listing an Installed MIDlet Suite

The `listMidlets.sh` application is located in the *InstallDir*/midp/bin directory. To list an installed MIDlet suite under the Java programming language AMS, type the command as shown below. There are no arguments to the `listMidlets.sh` command.

```
$ listMidlets.sh
```

This returns a list of currently installed Midlet suites.

Note – If entering the `listMidlets.sh` command on a Linux platform using the bash shell, enter the following:

```
$ ./listMidlets.sh
```

Removing an Installed MIDlet Suite

The `removeMidlet.sh` application is located in the `InstallDir/midp/bin` directory. To remove a MIDlet suite under the Java programming language AMS, type the command as shown below.

```
$ removeMidlet.sh ( suiteID | all )
```

■ *suiteID*

The unique integer assigned to a MIDlet suite when it was installed. It must be provided for a specific MIDlet suite to be removed.

■ `all`

Requests that all MIDlet suites be removed.

Note – If entering the `removeMidlet.sh` command on a Linux platform using the bash shell, enter the following:

```
$ ./removeMidlet.sh ( suiteID | all )
```

Emulating the Graphical AMS of a MIDP Phone

Java Wireless Client software provides a graphical AMS that emulates the user interface on a MIDP phone. Use the `usertest` shell script located in the `installDir/midp/bin` directory to run the graphical application manager MIDlet. The `usertest` script takes no arguments. The following summarizes its command line:

```
$ usertest
```

Note – If entering the `usertest` command on a Linux platform using the bash shell, enter the following:

```
$ ./usertest
```

The `usertest` script interacts differently with the standard logging service than the other executables described in this chapter. The default implementation of the logging and tracing utility prints log data to `stdout`. As a result, the other executables (such as `autotest`, described in [Chapter 7](#)) display log or trace output on the terminal in which you started them. Alternatively, you can pipe the output to a file, as you would for any application.

Note – On Win32-based platforms, the command to use in place of `usertest` is `runMidlet`. For more information, see [“Using `runMidlet` on a Windows/i386 Platform”](#) on page 23.

Debugging a MIDlet

This chapter describes how to run a proxy and a debugger that use the K Virtual Machine (KVM) Debug Wireless Protocol (KDWP) so that you can debug a MIDlet. This chapter assumes that the MIDlet is packaged in a JAR file, and that the classes to be debugged are not ROMized.

The Debugging Process

1. Build the CLDC HotSpot™ Implementation with the build-time option

`ENABLE_JAVA_DEBUGGER=true.`

See the *Build Guide* for more information.

2. Build the Java Wireless Client software with the build-time option

`USE_JAVA_DEBUGGER=true.`

See the *Build Guide* for more information.

3. Ensure that your application is compiled with the `-g` option to `javac`, and preverify and package it as usual.

4. Install your application on the Java Wireless Client software using the `installMidlet` application.

See [“Using the OTA Installer” on page 21](#) for instructions on using the `installMidlet` application.

5. Start the KDP proxy.

See your CLDC documentation for more information. For example:

```
$ java -classpath CLDC_DIST_DIR/bin/kdp.jar kdp.KVMDebugProxy -l 1234
-p -r localhost 2808
```

If you are starting the proxy on a different machine from your CLDC Hotspot Implementation, substitute the CLDC host's name for `localhost`.

6. Run the MIDlet that needs to be debugged.

See [“Running an Installed MIDlet” on page 22](#) for more information on running a MIDlet. For example:

```
$ ./runMidlet_g -port 2808 -debug 1
```

Note that the port for runMidlet is the same as the one given to the KDP proxy.

7. Run the debugger, **jdb**.

Use the `-connect` option to provide the location of the KDP proxy. For example:

```
$ jdb -connect com.sun.jdi.SocketAttach:hostname=localhost,port=1234
```

See your CLDC documentation for more information.

Automatically Testing MIDlet Suite Management and Execution

Java Wireless Client provides two scripts that automatically install, run, and remove one or more MIDlet suites. The `autotest` script works with a single MIDlet suite, and the `autotestm` script works with multiple MIDlet suites. This chapter covers both scripts.

autotest Script

The `autotest` script runs the `com.sun.midp.installer.AutoTester` MIDlet. The MIDlet installs a MIDlet suite from a given location, runs the suite in a loop for a specified number of iterations or until a new version of the suite is not found, then removes the suite. The `autotest` script is in the `InstallDir/midp/bin` directory.

The following summarizes the command line for the `autotest` script:

```
autotest [url [securityDomain:permissions] [numberOfIterations]]
```

- *url*

The HTTP or HTTPS URL of a JAD or JAR file. The MIDlet suite is installed from this URL. If this argument is not given, a form queries the user for the arguments.

- *securityDomain*

The security domain to use for unsigned MIDlet suites. If provided, this value overrides the default domain for unsigned suites, which is called `unidentified`.

- *numberOfIterations*

Integer specifying how many times to run the suite. If this argument is not given or is less than zero, the script executes the suite and then attempts to install the next suite using the same URL. If this attempt is successful, the next suite is executed. Otherwise, the script exits.

- *permissions*

On optional parameter, this specifies a list of permissions that must be allowed for *securityDomain*. The all setting grants permissions to all users.

autotestm Script

Use the autotestm script to run the MIDlet `com.sun.midp.installer.AutoTesterMulti`. The MIDlet fetches a list of URLs that point to the suites to install. In parallel for each specified suite, the MIDlet installs or updates the suite, runs the first MIDlet in the suite in a loop for either the specified number of iterations or until the new version of the suite is not found, then removes the suite.

The autotestm script is in the `InstallDir/midp/bin` directory. It is only available in MVM mode (that is, if Java Wireless Client was built with `USE_MULTIPLE_ISOLATES=true` and CLDC HI was built with `ENABLE_ISOLATES=true` option).

The following summarizes the command line for the autotestm script:

```
autotestm [url [securityDomain:permissions] [numberOfIterations]]
```

- *url*

The HTTP or HTTPS URL of an HTML page with links to MIDlet suites. The HTML page must have lines with this form in the body:

```
<a href="http://machine[:port]/suiteName.jad">suiteName</a>
```

For example:

```
<a href="http://localhost/suite1.jad">Suite1</a>
```

```
<a href="http://localhost/suite2.jad">Suite2</a>
```

If *url* is not given, a form queries the user for the autotest script's arguments.

- *securityDomain*

The security domain to use for unsigned MIDlet suites. If provided, this value overrides the default domain for unsigned suites, which is called `unidentified`.

- *numberOfIterations*

Integer specifying how many iterations to run the suite. If this argument is not given or is less than zero, the script executes the suite and then attempts to install the next suite using the same URL. If this attempt is successful, the next suite is executed. Otherwise, the script exits.

Running i3 Tests

Use the `i3test` script to run the `i3test` MIDlet, which is in the package `com.sun.midp.i3test.Framework`. The MIDlet lists known i3 tests and runs either a specified test or all of the tests.

The `i3test` script is in the `installDir/midp/bin` directory. It is available only if Java Wireless Client was built with `USE_I3_TEST=true`.

The following summarizes the command line for the `i3test` script:

```
i3test [-list] [-selftest] [-verbose] [testclass]
```

- `-list`
Prints a list of known tests.
- `-selftest`
Runs the framework's self test.
- `-verbose`
Enables verbose output.
- *testclass*
The class name of the test to run. If this argument is not given, the script runs all known tests.

A

Upgrading Configuration Files

To easily migrate to Sun Java Wireless Client software 2.1 from earlier versions (Sun Java Wireless Client software, versions 1.1.2 and before), some upgrading of configuration files is needed. Two utilities provided by Sun Java Wireless Client, version 2.1 make it easy to do these upgrades.

Note – If you are currently working with Java Wireless Client software, version 1.1.3, no migration is needed to install and run Java Wireless Client software, version 2.1. This appendix applies only to Sun Java Wireless Client software, versions 1.1.2 and before.

Upgrade Localization Strings Tool

The UpgradeL10N tool upgrades a `LocalizedStrings<locale>.java` file from Java Wireless Client software, version 1.1.2 to the XML file format supported by Java Wireless Client software, version 2.1. The following is an example file:

```
InstallDir/src/configuration/configuration_xml/share/l10n/en-US.xml
```

It is included in the Java Wireless Client software source bundle and generated by the UpgradeL10N tool for the following file, found in the Java Wireless Client software, version 1.1.2, source bundle.

```
InstallDir/src/i18n/common/classes/com/sun/midp/l10n/LocalizedStrings.java
```

The tool is available in the folder `tool/upgradel10n`. It works on the desktop and should be manually compiled with the Java technology-based compiler from JDK software, version 1.4.1_01 or later.

Usage:

```
javac UpgradeL10N.java  
  
java -cp . UpgradeL10N [inencoding] [infile] \  
[outencoding] [outfile] [outclass]
```

You must specify an [inencoding] that is the same as the character encoding used by [infile].

For example:

```
java -cp . UpgradeL10N ISO-8859-1 LocalizedStrings.java \  
ISO-8859-1 en-US.xml LocalizedStringsBase
```

UpgradeROMizedProperties Tool

The Adaptive User Interface (AUI) Technology software (formerly known as Project Chameleon) enables extensive visual customization and visual skinning. Many skinning and customization properties are used inside the Java Wireless Client software. In the Java Wireless Client software, version 1.1.2 these properties were hard coded in Java programming language sources, while the Java Wireless Client software uses a flexible and easy maintainable XML representation of the AUI internal properties.

The `UpgradeRomizedProperties` tool converts the Chameleon properties stored in `RomizedProperties.java` included in the Java Wireless Client software, version 1.1.2 source bundle to the XML representation introduced in the Java Wireless Client software, version 1.1.3, and used in the Java Wireless Client software.

In the Java Wireless Client software 1.1.2, the property values are indexed using property names (strings) and therefore stored in a hash table internally by Chameleon. In the Java Wireless Client software, the Chameleon properties are presented in the XML format for further processing by the Configurator tool. The Configurator tool generates source code to store the Chameleon properties values in arrays indexed by integer constants.

Follow this syntax to run the conversion:

```
javac UpgradeRomizedProperties.java  
  
java -cp . UpgradeRomizedProperties infile outfile
```

infile is the path to `RomizedProperties.java` to convert.

outfile is the name of the XML data file with Chameleon properties.

Glossary

- API** Application Programming Interface. A set of classes used by programmers to write applications, which provide standard methods and interfaces and eliminate the need for programmers to reinvent commonly used code.
- AMS** Application Management Service. The system functionality that completes tasks such as installing applications, updating applications, and switching foregrounds.
- Application list** The screen that lists all of the installed applications. The user gets to this screen by pressing the `Apps` soft key on the home screen. The application list uses text color to show which applications are running. It also provides a system menu that enables the user to perform application management tasks on the highlighted application.
- Background** An application state in which the application does not receive events from its input stream and its displayable is not rendered to the screen.
- CDC** Connected Device Configuration. A Java ME platform configuration for devices, it requires a minimum of 2 megabytes of memory and a network connection that is always on.
- CLDC** Connected Limited Device Configuration. A Java ME platform configuration for devices with less than 512 kilobytes of RAM and an intermittent (limited) network connection, it uses a stripped-down Java virtual machine called the KVM, as well as several minimalist Java platform APIs for application services.
- Configuration** Defines the minimum Java runtime environment (for example, the combination of a Java virtual machine and a core set of Java platform APIs) for a family of Java ME platform devices.
- Foreground** The application state in which the application is rendered to the device display and the input stream is passed to it.
- Foreground switching** Changing which application is in the foreground by shifting the focus from one application to another.
- GCF** Generic Connection Framework. A part of CLDC, it improves network connectivity for wireless devices.

Home screen	The main screen of the application manager. This is the screen the user sees after they exit an application.
HTTP	HyperText Transfer Protocol. The most commonly used Internet protocol, based on TCP/IP, which is used to fetch documents and other hypertext objects from remote hosts.
HTTPS	Secure HyperText Transfer Protocol. A protocol for transferring encrypted hypertext data using Secure Socket Layer (SSL) technology.
JAD file	Java Application Descriptor file. A file provided in a MIDlet suite that contains attributes used by application management software (AMS) to manage the MIDlet's life cycle, as well as other application-specific attributes used by the MIDlet suite itself.
JAR file	Java Archive file. A platform-independent file format that aggregates many files into one. Multiple applications written in the Java programming language and their required components (.class files, images, sounds, and other resource files) can be bundled in a JAR file and provided as part of a MIDlet suite.
Java Community Process™ (JCP™) program	Java Community Process program. An open organization of international developers and licensees who develop and revise Java platform specifications, reference implementations, and technology compatibility kits using a formal submission and approval process.
Java ME platform	Java Platform, Micro Edition. A group of specifications and technologies that pertain to running the Java platform on small devices, such as cell phones, pagers, PDAs, and set-top boxes. More specifically, the Java ME platform consists of a configuration (such as CLDC or CDC) and a profile (such as MIDP or Personal Basis Profile) tailored to a specific class of device.
Java Specification Request (JSR)	A proposal for developing new Java platform technology, which is reviewed, developed, and finalized into a formal specification by the JCP program.
Java Virtual Machine	A software “execution engine” that safely and compatibly executes the byte codes in Java class files on a microprocessor.
KVM	A Java virtual machine designed to run in small devices, such as cell phones and pagers. The CLDC configuration is designed to run in a KVM.
LCD	Liquid Crystal Display. A common kind of screen display often used in small devices.
LCDUI	Liquid Crystal Display User Interface. A user interface toolkit for interacting with LCD screens in small devices. More generally, a shorthand way of referring to the MIDP user interface APIs.
MIDlet	An application written for MIDP.

MIDlet suite	A way of packaging one or more midlets for easy distribution and use. Each MIDlet suite contains a Java application descriptor file (.jad), which lists the class names and files names for each MIDlet, and a Java Archive file (.jar), which contains the class files and resource files for each MIDlet.
MIDP	Mobile Information Device Profile. A specification for a Java ME platform profile, running on top of a CLDC configuration, which provides APIs for application life cycle, user interface, networking, and persistent storage in small devices.
Obfuscation	A technique used to complicate code by making it harder to understand when it is de-compiled. Obfuscation makes it harder to reverse-engineer applications and therefore, steal them.
Optional Package	A set of Java ME platform APIs that provides additional functionality by extending the runtime capabilities of an existing configuration and profile.
PNG	Portable Network Graphics. An image format commonly used with MIDP that can be compressed, transmitted, and stored without losing image quality.
Preemption	Taking a resource, such as the foreground, from another application.
Preverification	Due to limited memory and processing power on small devices, the process of verifying Java technology classes is split into two parts. The first part is preverification and done off-device using the preverify tool. The second part, which is verification, is done on the device at runtime.
Profile	A set of APIs added to a configuration to support specific uses of a mobile device. Along with its underlying configuration, a profile defines a complete and self-contained application environment.
Provisioning	A mechanism for providing services, data, or both to a mobile device over a network.
Push Registry	The list of inbound connections, across which entities can push data, maintained by the Java Wireless Client software. Each item in the list contains the URL (protocol, host, and port) for the connection, the entity permitted to push data through the connection, and the application that receives the connection.
RMI	Remote Method Invocation. A feature of Java SE technology that enables Java technology objects running in one virtual machine to seamlessly invoke objects running in another virtual machine.
RMS	Record Management System. A simple record-oriented database that enables a MIDlet to persistently store information and retrieve it later. MIDlets can also use the RMS to share data.
SMS	Short Message Service. A protocol allowing transmission of short text-based messages over a wireless network.

SOAP Simple Object Access Protocol. An XML-based protocol that allows objects of any type to communicate in a distributed environment, it is most commonly used to develop web services.

SSL Secure Sockets Layer. A protocol for transmitting data over the Internet using encryption and authentication, including the use of digital certificates and both public and private keys.

Sun Java Device Test

Suite A set of Java programming language tests developed specifically for the wireless marketplace, providing targeted, standardized testing for CLDC and MIDP on small and handheld devices.

SVM Single Virtual Machine. A mode of the Java Wireless Client software, it can run only one MIDlet at a time.

task At the platform level, each separate application that runs within a single Java virtual machine is called a task. The API used to instantiate each task is a stripped-down version of the Isolate API defined in JSR 121. See the *CLDC HotSpot Implementation Architecture Guide* for more information.

TCP/IP Transmission Control Protocol/Internet Protocol. A fundamental Internet protocol that provides for reliable delivery of streams of data from one host to another.

WAE Wireless Application Environment. It provides an application framework for small devices, by leveraging other technologies such as WAP, WTP, and WSP.

WAP Wireless Application Protocol. A protocol for transmitting data between a server and a client (such as a cell phone) over a wireless network. WAP in the wireless world is analogous to HTTP in the World Wide Web.

WMA Wireless Messaging API. A set of classes for sending and receiving Short Message Service messages.

(x) button The button the user presses to end a task. On a real device this is the End key. On Windows it is the End key and sometimes the power key on the phone skin.

Index

Symbols

`_main.ks` file, 2

A

adding keys to ME keystores, 4

applications

`listMidlets`, 25

`removeMidlet`, 25

attributes

 MIDlet-Certificate-*m-n*, 13, 14

 MIDlet-Jar-RSA-SHA1, 11, 14

 MIDlet-Jar-URL, 14

autotest script, 31

AutoTester MIDlet, 31

AutoTesterMulti MIDlet, 32

autotestm script, 32

B

build-time options

 ENABLE_JAVA_DEBUGGER, 29

 USE_JAVA_DEBUGGER, 29

C

certificate authority keys

 see public keys

certificate chain, 11

creating alternate ME keystores, 3

D

debugging MIDlet suites, 29

deleting keys from ME keystores, 6

digital signature, 12, 14

domains, security, 4, 5

E

EMSA-PKCS1-v1_5 encoding, 14

emulating a graphical AMS, 27

ENABLE_JAVA_DEBUGGER build-time option, 29

encodings

 EMSA-PKCS1-v1_5, 14

 UTF-8, 13, 14

F

files

`j2se_test_keystore.bin`, 10

`JadTool.jar`, 10

`listMidlets.c`, 25

`_main.ks`, 2

`MEKeyTool.jar`, 2

`removeMidlet.c`, 25

G

graphical AMS, emulating, 27

graphical application manager MIDlet, 27

I

`i3test` MIDlet, 33

`i3test` script, 33

importing public keys, 4

installing MIDlet suites, 21

`installMidlet` script, 21

intermediate certificates, 11

- J**
- j2se_test_keystore.bin file, 10
 - JadTool utility
 - instructions, 9
 - options, 13
 - JadTool.jar file, 10
 - JCA keystores, 2
 - jdb, 30
- K**
- KDP proxy, 29
 - KDWP, 29
 - keys
 - see public keys
 - keystores, 2
 - keytool utility, 1, 10
- L**
- listing keys in ME keystores, 5
 - listing MIDlet suites, 25
 - listMidlets application, 25
 - listMidlets.c file, 25
 - logging service, 28
- M**
- managing
 - alternate ME keystores, 3
 - public keys, 8
 - ME keystores, 2
 - adding keys to, 4
 - creating alternates, 3
 - deleting keys from, 6
 - importing keys into, 4
 - listing keys in, 5
 - managing alternate, 3
 - replacing keys in, 7
 - using alternate, 4
 - MEKeyTool
 - instructions, 2
 - options, 8
 - MEKeyTool.jar file, 2
 - MIDlet suites
 - automatic testing, 31
 - debugging, 29
 - installing, 21
 - listing, 25
 - running, 22
 - signing, 9
 - MIDlet-Certificate-*m-n* attribute, 13, 14
 - MIDlet-Jar-RSA-SHA1 attribute, 11, 14
 - MIDlet-Jar-URL attribute, 14
 - MIDlets
 - AutoTester, 31
 - AutoTesterMulti, 32
 - graphical application manager, 27
 - i3test, 33
 - OTA installer, 21
- O**
- OTA Installer, 21
 - OTA installer MIDlet, 21
- P**
- public keys
 - adding, 4
 - deleting, 6
 - importing, 4
 - listing, 5
 - replacing, 7
- R**
- removeMidlet application, 25
 - removeMidlet.c file, 25
 - replacing keys in ME keystores, 7
 - running MIDlet suites, 22
- S**
- scripts
 - autotest, 31
 - autotestm, 32
 - i3test, 33
 - installMidlet, 21
 - usertest, 27
 - security domains, 4, 5
 - signing MIDlet suites, 9
- T**
- testing MIDlet suites, automatic, 31
- U**
- USE_JAVA_DEBUGGER build-time option, 29
 - usertest script, 27

using alternate ME keystores, 4

UTF-8 encoding, 13, 14

utilities

JadTool, 9

keytool, 1

