Unified Emulator Interface Specification

Version 1.0.2

April 2006

Copyright Notice

License

LIMITED LICENSE GRANTS

1. License Grant. Licensor hereby grants you a perpetual, nonexclusive, nontransferable, worldwide, fully paidup, royalty free, limited license (without the right to sublicense) under any applicable copyrights or, subject to the provisions of subsection 2 below, patent rights it may have covering the Specification to:

a. display, perform and copy the Specification;

b. create and distribute implementations, in whole or in part, of the Specification; and

c. distribute the Specification, unmodified, to third parties provided that you pass through and make any further distribution of the UEI Specification subject to this license. No license is granted hereunder for any other purpose (including, for example, modifying the Specification, other than to the extent of your fair use rights). Also, no right, title, or interest in or to any trademarks, service marks, or trade names of Licensor or Licensor's licensors,

Licensor or the Licensor's licensors is granted hereunder.

2. Patent Rights.

a. Licensor covenants that, subject solely to the reciprocity requirement described below, it will not seek to enforce any of its Necessary Claims against any compliant implementation of the UEI Specification, where “compliant” is understood to be determined through actual functionality as evidenced by the implementation in question, whether or not validated by any SelfServ Tests. Notwithstanding the commitment above, this covenant shall not apply, and Licensor makes no assurance, covenant or commitment not to assert or enforce any or all of its patent rights against any individual, corporation or other entity that asserts, threatens or seeks at any time to enforce its own or another party's Necessary Claims against any compliant implementation of the UEI Specification.

b. This covenant is not an assurance either (i) that any of Licensee's issued patents cover an UEI implementation or are enforceable, or (ii) that an UEI implementation would not infringe patents or other intellectual property rights of any third party.

c. For the purposes of this Section 2, “Necessary Claims” means those claims of all patents, pending patent applications and utility models, regardless of when issued or effective, which are enforceable by Licensor (including by subsidiaries under its control) and which are necessarily infringed by a compliant implementation of the UEI Specification as approved by the UEI Board, where such infringement could not have been avoided by another technically feasible noninfringing implementation of the UEI Specification. Notwithstanding the foregoing sentence, Necessary Claims do not include any claims other than those set forth above even if contained in the same patent as Necessary Claims. Similarly, Necessary Claims do not include any claims enforceable against elements or features of a product which do not themselves implement portions of the UEI Specification, even where such elements or features are included in a product which also includes an implementation of the UEI Specification.

DISCLAIMER OF WARRANTIES

THE SPECIFICATION IS PROVIDED "AS IS". Licensor MAKES NO REPRESENTATIONS OR WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT (INCLUDING AS A CONSEQUENCE OF ANY PRACTICE OR IMPLEMENTATION OF THE SPECIFICATION), OR THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE. This document does not represent any commitment to release or implement any portion of the Specification in any product. In addition, the Specification could include technical inaccuracies or typographical errors.

LIMITATION OF LIABILITY

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL Licensor OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUE, PROFITS OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED IN ANY WAY TO YOUR HAVING, IMPELEMENTING OR OTHERWISE USING USING THE SPECIFICATION, EVEN IF Licensor AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You will hold Licensor harmless from any claims arising or resulting from your use of the UEI Specification, including your distribution of any implementation of the UEI Specification.

MISCELLANEOUS

Restricted Rights Legend. U.S. Government: If this Specification is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.72024 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for nonDoD acquisitions).

Governing Law. Any action relating to or arising out of this Agreement will be governed by California law and controlling U.S. federal law. The U.N. Convention for the International Sale of Goods and the choice of law rules of any jurisdiction will not apply.

Export Control. As further described at http://www.sun.com/its, You agree to comply with applicable U.S. Export controls and trade laws of other countries that apply to the UEI Specification.

Integration. This License represents the complete agreement of the parties concerning the subject matter hereof.

Table of Contents

1 Introduction

The Unified Emulator Interface (UEI) is a standard for interaction between Integrated Development Environments (IDEs) and device emulators. IDE vendors who implement the UEI specification know that their products will work with a wide variety of device emulators. Device manufacturers who implement the UEI specification in an emulator are assured that their emulator will work with a wide variety of development tools. Developers are happy because their tools and emulators are interoperable. Customers win because the UEI simplifies the process of creating applications.

This document describes the requirements that an emulator must meet to be compliant with version 1.0.1 of the UEI.

Throughout this specification, the following definitions are used:

- REQUIRED means that the feature must be implemented.
- OPTIONAL means that the feature is not required to be implemented. However, if it is implemented, the implementation must fully meet the specification of the feature.

Unless noted otherwise, every part of the specification is required to be implemented. Optional features are noted as such.

NOTE: In this document, executable files have the extension `.exe`. This is only the case in Microsoft Windows environments. On UNIX® and Linux systems, these files are understood as not having the `.exe` extension. Therefore, any test for the presence or absence of these files must test for the correct file name.

This specification contains references to other documents:

- The CLDC specification includes a description of preverifying class files.

  `http://jcp.org/en/jsr/detail?id=139`

- The Java Debug Wire Protocol (JDWP) describes a standard interface for debugging applications.

  `http://java.sun.com/j2se/1.5.0/docs/guide/jpda/jdwp-spec.html`

2 Directory Structure

The emulator must be entirely contained within a single directory.

Inside the emulator directory, three directories are required. Additional directories can be present, but the following are required.

Table 1 Directory Structure

Directory	Description
`bin`	Contains commands. The next section describes the required commands.
`lib`	Contains profile and configuration API class file archives. By default, all archives (files with `.jar` or `.zip` extensions) in the `lib` directory must be used for MIDlet compilation and preverification. Subdirectories under `lib` must not be included. This default behavior can be modified with the `bootclasspath` key returned by `-Xquery`. See the section Query Format for more information. JAR files in `lib` should contain manifest files in the format described in the section API Manifests. **NOTE:** For the default behavior to work, emulators must archive the API classfiles. If `.class` files must be used outside of an archive, the `bootclasspath` key must be properly set in the `-Xquery` response.
`docs`	Contains API documentation generated by the Javadoc™ tool. The root of the documentation tree must be `docs/index.html`.

3 Commands

The following table contains a list of the commands defined by the UEI. The commands are located in the `bin` directory.

Table 2 Commands

Command Name	Description
`emulator.exe`	Runs the emulator. The emulator must be a MIDP implementation. Any textual output generated by the emulator (for example, from the `-Xquery` or `-Xverbose` command-line arguments, as well as output from `System.out.println`) must be made to standard output if possible. This allows for programmatic parsing of the output. For UEI implementations providing on-device execution, this might not always be possible. In general, UEI implementations must attempt to minimize the number of windows used during execution. The syntax of the `emulator` command is described in Emulator Execution.
`preverify.exe`	(Optional) Runs the preverifier, which prepares class files to be run on a device. The `preverify` command is required for emulators based on CLDC. The syntax of `preverify` is described in Preverifier Execution.
`prefs.exe`	(Optional) Runs the emulator configuration utility. This can be used to configure the emulator behavior. Because many IDE developers prefer to maintain a common look and feel, this command is optional. Provide this command with the emulator to assist in device

	configuration. Each emulator has one `prefs` command. **NOTE:** Changes made in `prefs` can affect the output of the emulator's `-Xquery` option. Therefore, any IDE using `-Xquery` must re-run and re-parse the `-Xquery` result after running the `prefs` command. The `prefs` command requires no command-line parameters.
`utils.exe`	(Optional) Supports any additional utilities provided with the emulator. For example, Sun's Java Wireless Toolkit implements the HTTPS protocol. To support this protocol, Sun might include an HTTPS certificate viewer and editor with its emulator. Each emulator has one `utils` command. The `utils` command requires no command-line parameters.
`mekeytool.exe`	(Optional) Manipulates the emulator's keystore. The syntax for this command is described in Key Tool Execution.

4 Preverifier Execution

The `preverify` command is required to accept the syntax described in this section. The preverifier performs bytecode preverification on class files as defined in the CLDC specification.

The following command-line options must be accepted by `preverify`.

Table 3 Preverifier Arguments

Option	Description
`-classpath` *list-of-directories*	Contains a list of locations containing classes that are referenced by the classes being preverified. Separate the list of directories and archives with semicolons (;) on Microsoft Windows systems and colons (:) on UNIX and Linux systems.
`-d` *output-directory*	Directory for preverified output class files. Output files must be created with their complete package tree. For example, write a preverified class `com.example.Main` to `com/example/Main.class`.

`-cldc`
 	
`-cldc1.0`
	Check preverified classes for use of language features that are not part of the CLDC 1.0 specification (for example, native methods, floating point, and finalizers) and ensure that the preverified classes are in the correct format for CLDC 1.0 virtual machines.
	
	If classes are being preverified to run on CLDC 1.0 devices, this option must be used. To preverify classes for CLDC 1.1 virtual machines, use the `-nofinalize` and `-nonative` options.
	
	The flags `-cldc` and `-cldc1.0` are equivalent. If the emulator declares (in response to `emulator -version`) that it only suports CLDC version 1.0, then `preverify` need not accept the `-cldc1.0` parameter, and an IDE must not pass it this parameter.

`-nofinalize`	Checks that finalizers are not used in preverified classes.
`-nonative`	Checks that native methods are not used in preverified classes.
`-nofp`	Checks that floating point operations are not used in preverified classes.
class-and-directory names	A list of the following: • Class files to be preverified • Directories containing class files to be preverified. Separate the list with spaces.
`@`*file-path*	Read command-line arguments from the specified file. Place all arguments on a single line and enclose directory names with double quotes (“).

For example, the following preverifies the classes in `tmpclasses` for CLDC 1.0, using the APIs in `midpapi.zip`:

```
preverify -cldc -classpath
C:\WTK23\lib\midpapi20.jar;C:\WTK23\lib\cldcapi10.jar -d classes
tmpclasses
```

The following preverifies the classes in `tmpclasse`s for CLDC 1.1, using the APIs in `midpapi.zip` and `wma.zip`:

```
preverify -nofinalize -nonative -classpath
C:\WTK23\lib\midpapi20.jar;C:\WTK23\lib\wma20.jar;C:\WTK23\lib\cldca
pi11.jar -d classes tmpclasses
```

5 Getting Information About the Emulator

The `emulator` command must provide information about itself upon request. This section describes the mechanisms by which the emulator supplies this information.

5.1 Emulator Information Arguments

The emulator command must support several arguments that provide information about the emulator itself.

Table 4 Emulator Information Arguments

`-version`	Display version information about the emulator. The first line is the emulator name and version. Subsequent lines are key and value pairs separated by a single colon and a single space. The valid keys and their values are as follows:
	• `Configuration:` *Configuration-Name*`[-`*Version*`]`
	• `Profile:` *Profile-Name*`[-`*Version*`]`
	• `Optional:` *Optional-API-Names*
	Optional-API-Names is a comma-separated list of optional API names and versions. If an API name has an associated version, the name and version are separated by a hyphen (`-`).
	For example:
	`Sun Java Wireless Toolkit 2.3 Beta2`
	`Profile: MIDP-2.0`
	`Configuration: CLDC-1.1`
	`Optional: J2ME-WS-1.0,J2ME-XMLRPC-1.0,JSR179-1.0,JSR180-1.0,JSR184-1.0,JSR211-1.0,JSR226-1.0,JSR229-1.0,JSR238-1.0,JSR239-1.0,JSR75-1.0,JSR82-1.0,MMAPI-1.1,SATSA-APDU-1.0,SATSA-CRYPTO-1.0,SATSA-JCRMI-1.0,SATSA-PKI-1.0,WMA-1.1,WMA-2.0`
	The configuration, profile, and API names and versions must match the names and versions of APIs in API JAR file manifests, if names and versions are specified there. See API Manifests for details.

`-Xquery`	Print emulator device information to the standard output and immediately exit. Printed information includes, but is not limited to, device names, device screen size, and other device capabilities. It

	is possible to get information for a single device by using the `-Xdevice` argument in conjunction with `-Xquery`. See the section **Query Format** for a description of the output from this option.
`-help`	Display a list of valid arguments in human-readable form.

Some overlap exists between the output of `emulator -version` and the *`device-name`*`.apis` properties returned from `emulator -Xquery`. The output of `emulator -version` describes the generic behavior of the emulator using a typical configuration. The output of `emulator -Xquery` describes in more detail the support provided by each device. It is possible for a device to declare through the response to `-emulator -Xquery` that it supports a different set of APIs than that described by `emulator -version`.

5.2 Return Code

If the execution of a MIDlet fails due to an uncaught exception, `emulator` must return a non-zero return code.

5.3 Query Output

When the emulator is run with the `-Xquery` option, information describing the emulator and its capabilities are sent to standard output. The general format is that of a properties file. Each property line can be defined with this simplified BNF:

```
property-line ::= comment | property
property ::= key ':' value end-of-line

end-of-line ::= '\n' | '\r\n'

comment ::= '#' characters end-of-line
```

A `key` can contain any character other than a whitespace character or a colon (`:`). The `value` can contain internal whitespace characters. However, it might also contain any of the standard escaped characters: `\t`, `\n`, `\r`, `\\`, `\"`, `\'`, and `\`*`space`* (a backslash followed by a space). If `value` is blank, the property defined by `key` is an empty string.

The properties returned apply to either all devices in the emulator or to a specific device. The properties for all devices are listed in Table 5.

Table 5 Properties For All Devices

Name	Value
`uei.version`	(Optional) The version of the UEI specification supported by this emulator. The value must be either `1.0` or `1.0.1`. If this property is not specified, its value is `1.0`.
`uei.arguments`	(Optional) A list of the UEI arguments supported by all devices. The format of this list is described after this table. If this property is not specified, its value is the following:

	D,Xverbose,Xquery,Xdebug,Xrunjdwp,Xdevice,Xdescriptor,Xjam This list can be overridden for a specific device, as described later.
device.list	The value for device.list is a comma-separated list of device names. Even if an emulator supports only one device, this property must still be produced.
security.domains	(Optional) A comma-separated list of security domains supported by the emulator. This property is required if the mekeytool command is present. This list can be overridden for a specific device, as described later.

The uei.arguments list can include arguments as well as specific sub-commands. Separate the arguments and sub-commands in the list with commas, using no spaces. If an argument allows for a sub-command (for example, the -Xverbose argument), the sub-command must be listed after the main argument following a colon. As a shorthand notation, sub-commands do not need to be listed if all sub-commands are supported. However, if only a subset of sub-commands is supported, every supported sub-command must be explicitly listed. For example, if all -Xjam sub-commands are supported, but only the allocation and class -Xverbose sub-commands are supported, uei.arguments line includes the following:

```
uei.arguments: Xjam,Xverbose:allocation,Xverbose:class, ...
```

The device-specific properties start with the device name. The device name must be one of the values from the device.list property.

Table 6 Device-Specific Properties

Name	Value
device-name.screen.width	The width of the device screen in pixels.
device-name.screen.height	The height of the device screen in pixels.
device-name.screen.isColor	true for color screens, false for grayscale.
device-name.screen.bitDepth	The number of bits that describe a pixel's color. For grayscale screens, this property describes how many bits are used to determine the level of gray.
device-name.screen.isTouch	true for screens that support pointer events, false otherwise
device-name.uei.arguments	(Optional) Has the same meaning and syntax as uei.arguments, but it applies to a specific device and overrides the value of uei.arguments.
device-name.bootclasspath	A list of API files that must be used to build a MIDlet that runs on this device. The format is a comma-separated list of fully qualified pathnames and/or filenames. All paths or files must start with a drive specifier[1]. A path or file might contain internal spaces, but must never contain a comma. Trailing spaces are ignored.

1. A drive specifier is not required if the emulator is running on a UNIX platform. However, for the UNIX platform, the path or file must start with the root directory.

	This property may contain files (`.zip`, `.jar`, or `.class`), directories, or both. Paths must use a slash (/) as the directory separator. The application processing this value might need to transform the slash (/) into the proper directory separator for the platform on which it is running.
device-name`.apis`	(Optional) Contains the same information, in the same format, as *device-name*`.bootclasspath` and also includes any earlier versions of APIs supported by this device. This enables an IDE to select which versions of the API to use to build an application. If this property is not present, its value is the same as *device-name*`.bootclasspath`.

device-name`.version.configuration`	(Optional) The configuration supported by this device. A device can support only one configuration. Emulators that support multiple configurations can do so by providing separate devices for each. Acceptable configuration names are:
	
	`CDLC-1.0`
	`CDLC-1.1`
	`CDC-1.0`
	`CDC-1.1`

device-name`.version.profile`	(Optional) A list of one or more profiles supported by this device. Multiple items are separated by commas. Acceptable profile names are:
	
	`MIDP-1.0`
	`MIDP-2.0`
	`IMP-1.0`
	`IMP-NG`

device-name`.security.domains`	(Optional) Has the same meaning and syntax as `security.domains`, but it applies to a specific device and overrides the value of `security.domains`.

Emulators are free to return additional properties in response to `-Xquery`. IDEs must ignore any properties that do not interest them.

An example response from the `emulator -Xquery` command follows:

```
# List of supported devices
device.list: DefaultColorPhone
uei.version: 1.0.1
uei.arguments:
    Xverbose,Xquery,Xdebug,Xrunjdwp,Xdevice,Xdescriptor,Xjam,Xautotest,Xheapsize

# Properties for device DefaultColorPhone
DefaultColorPhone.description: DefaultColorPhone
DefaultColorPhone.screen.width: 240
DefaultColorPhone.screen.height: 320
DefaultColorPhone.screen.isColor: true
DefaultColorPhone.screen.isTouch: false
DefaultColorPhone.screen.bitDepth: 8
```

```
DefaultColorPhone.bootclasspath:
   C:/WTK23/lib/midpapi20.jar,C:/WTK23/lib/cldcapi11.jar,C:/WTK23/lib/wma20.jar,C:/W
   TK23/lib/mmapi.jar,C:/WTK23/lib/j2me-ws.jar,C:/WTK23/lib/j2me-
   xmlrpc.jar,C:/WTK23/lib/jsr75.jar,C:/WTK23/lib/jsr082.jar,C:/WTK23/lib/jsr184.jar
   ,C:/WTK23/lib/jsr179.jar,C:/WTK23/lib/satsa-apdu.jar,C:/WTK23/lib/satsa-
   pki.jar,C:/WTK23/lib/satsa-
   crypto.jar,C:/WTK23/lib/jsr211.jar,C:/WTK23/lib/jsr238.jar,C:/WTK23/lib/jsr229.ja
   r,C:/WTK23/lib/jsr180.jar,C:/WTK23/lib/jsr234.jar,C:/WTK23/lib/jsr226.jar,C:/WTK2
   3/lib/jsr239.jar
DefaultColorPhone.stub.classpath: C:/WTK23/wtklib/emptyapi.zip
DefaultColorPhone.apis:
   C:/WTK23/lib/cldcapi10.jar,C:/WTK23/lib/cldcapi11.jar,C:/WTK23/lib/midpapi10.jar,
   C:/WTK23/lib/midpapi20.jar,C:/WTK23/lib/wma20.jar,C:/WTK23/lib/wma11.jar,C:/WTK23
   /lib/mmapi.jar,C:/WTK23/lib/jsr75.jar,C:/WTK23/lib/jsr082.jar,C:/WTK23/lib/jsr184
   .jar,C:/WTK23/lib/jsr179.jar,C:/WTK23/lib/jsr211.jar,C:/WTK23/lib/jsr238.jar,C:/W
   TK23/lib/jsr229.jar,C:/WTK23/lib/jsr180.jar,C:/WTK23/lib/jsr234.jar,C:/WTK23/lib/
   j2me-ws.jar,C:/WTK23/lib/j2me-
   xmlrpc.jar,C:/WTK23/lib/jsr226.jar,C:/WTK23/lib/satsa-
   apdu.jar,C:/WTK23/lib/satsa-jcrmi.jar,C:/WTK23/lib/satsa-
   pki.jar,C:/WTK23/lib/satsa-crypto.jar,C:/WTK23/lib/jsr239.jar
DefaultColorPhone.security.domains:
   manufacturer,minimum,identified_third_party,unidentified_third_party,maximum
```

6 Running Local Applications

An emulator can be run in one of two modes. The first, and most common for development, is running a MIDlet directly from classes in the file system. The second mode, which is optional, is to run the emulator according to the Over The Air (OTA) User Initiated Provisioning Recommended Practice document, which is part of the MIDP 2.0 specification. This section describes running applications on the emulator from the local file system.

The command line syntax is simple:

```
emulator [arguments] [MIDlet class name]
```

Two supported methods specify emulator parameters. One is the `prefs` command. The other is the command-line interface. If both mechanisms are used, the command-line interface takes higher precedence. Other mechanisms, such as environment variables or a system registry, are not needed or required.

Table 7 lists the arguments for running local applications.

Table 7 Arguments For Running Local Applications

Argument	Description
`-classpath`	Set the classpath for the emulator. **NOTE:** A user of the UEI must never put API classes in the `-classpath` command-line option. This includes any classes automatically picked up from the `lib` directory or any classes specified in the `bootclasspath` key returned by the `-Xquery` command-line option. The restriction applies for both running and debugging the MIDlet.
`-D`*property=value*	(Optional) Set the system property *property* to *value* in the emulator. *property* can be any non-empty string. For example: `emulator -Dmyproperty=myvalue ...` This can be used during development to pass parameters to an application without rebuilding or repackaging it.
`-Xdescriptor:`*jad-file*	(Optional) Run an application using the specified Java Application Descriptor file.
`-Xdevice:`*device-name*	(Optional) Run an application on the device specified by *device-name*.
`-Xheapsize:`*size[*`k`|`M`*]*	(Optional) Set the emulator's heap size to be a maximum of *size* bytes. *size* can be written as a plain number to signify bytes. Append either a `k` or an `M` to signify kilobytes or megabytes, respectively.

For example, this command line runs the MIDlet `PianoMIDlet` from `wj2.jar`:

```
emulator -classpath wj2.jar PianoMIDlet
```

On an emulator with support for `-Xdescriptor`, this command line runs the MIDlet suite defined in `wj2.jad`:

```
emulator -Xdescriptor:wj2.jad
```

NOTE: Although running from the file system implies an emulator running on the host PC, that is not the only possibility. In fact, the implementor of the UEI could package the MIDlet class files and synchronize them to a real device for execution.

---

7 Running in OTA Mode

When running the emulator in OTA mode, no mechanism exists to run or install an application directly from a local file system. All MIDlet suites executed using OTA must be installed onto the

emulator or a device using OTA and HTTP.

Support for running via OTA is optional. An IDE cannot assume that running via OTA is implemented, unless the emulator returns `Xjam` in a response to `emulator -Xquery`.

The basic syntax is:

```
emulator -Xjam:<arguments>
```

For example:

```
emulator -Xjam:install=http://example.org/Application.jad
```

The interactive application manager can be invoked by writing `-Xjam` without any additional arguments, as follows:

```
emulator -Xjam
```

Additional arguments can be specified to instruct the emulator to perform specific actions, as detailed in the following table.

Table 8 Emulator OTA Arguments

OTA Extensions		Description
`-Xjam`		Run the interactive application manager.
`-Xjam:`*command*		Instruct the application manager to perform a specific action. The *application* is specified uniquely for each *command*. One *command* must always be given. There must be a system-defined application named `all`. The `all` application refers to all user installed applications and is only valid for specific commands. It is illegal to allow a user installed application to be named `all`.
	Command	**Description**
	`install=`*`url`*	Install an application onto the emulator from *url*, which must be a valid URL. HTTP URLs must be supported. An IDE must not assume that any other protocol is available for installing applications.
	`force`	Use `force` in conjunction with `install`. If an existing application has the same storage name as the application to be installed, `force` removes the existing application prior to installing the new application. For example: `-Xjam:install=`*`url`* `-Xjam:force`
	`run=`*`application`*	Run a previously installed application. The *application* parameter must be a valid storage name or the storage number of the MIDlet.
	`remove=`*`application`*	Remove a previously installed application. The *application* parameter must be a valid storage name or the storage number of the MIDlet. The system-defined application `all` can be used to remove all installed applications.
	`transient=`*`url`*	Install, run, and remove an application. The *url* parameter must be a valid URL pointing to the application's descriptor file. `transient` is a shortcut for launching the emulator three separate times to install, run, and remove the application.
	`list`	List all applications installed on the device and exit. The format of the output is described later. After writing the list of applications to standard output, `emulator` must exit.
	`storageNames`	List all applications installed on the device in a format easily parsed by another program. Each line contains one storage name in numerical order. Only the storage name is listed. The order of the list is important. The first storage name must be storage number `1`. After writing the list of applications to standard output, `emulator` must exit.

The `-Xdevice` and `-Xheapsize` arguments, presented previously, can be used in conjunction with `-Xjam`.

The output from `emulator -Xjam:list` is as follows:

Header-lines
Installed-MIDlet-Suite-data
Blank-line

Zero or more header lines can be present. Do not parse header information. Zero or more *Installed-MIDlet-Suite-data* blocks dan be present.

Each MIDlet suite is represented by an *Installed-MIDlet-Suite-data* block, which is defined as follows:

[*MIDlet-suite-number***]**
Name: *MIDlet-suite-name>*
Vendor: *Vendor-name*
Version: *MIDlet-suite-version*
Storage name: *Storage-name*
Size: *Application-size*
Installed From: *URL*
MIDlets:
MIDlet-name

MIDlet-Suite-number starts at 1 and counts up by 1 for each MIDlet Suite that is reported. The *Storage-name* for the MIDlet suite can be used as an argument to the `run` and `remove` commands of the `-Xjam` argument. *Application-size* can be expressed in bytes or kilobytes. Kilobytes are indicated by appending `K` to the number. One or more *MIDlet-name* lines can be present.

Starting from **`Name:`**, lines are indented by two spaces. Starting from the line after **`MIDlets:`**, lines are indented by four spaces.

Here is one example of the output from `-Xjam:list`:

```
C:\WTK23\bin>emulator -Xjam:list
Running with storage root DefaultColorPhone
Running with locale: English_United States.1252
[1]
  Name: JBricks
  Vendor: Sun Microsystems, Inc.
  Version: 1.0
  Authorized by: ST=state;L=city;O=org;OU=orgUnit;CN=cName
  Description: Test game for the Payment API
  Storage name: #Sun%0020#Microsystems%002c%0020#Inc%002e_#J#Bricks_
  Size: 34K
  Installed From: http://localhost:2697/JBricks/bin/JBricks.jad
  MIDlets:
    jbricks
[2]
  Name: WirelessJava
  Vendor: Jonathan Knudsen
  Version: 2.0
  Storage name: #Jonathan%0020#Knudsen_#Wireless#Java_
  Size: 79K
  Installed From: http://localhost/midp/bin/WirelessJava.jad
  MIDlets:
    PianoMIDlet
    Jargoneer
    StationSignMIDlet
    QuatschMIDlet
```

8 Debugging and Testing

The emulator can offer additional features for producing diagnostic output, enabling a debugger, and supporting automated testing. All of these features are optional; an IDE can determine support by using `-Xquery`. This section describes these optional arguments to the `emulator` command.

8.1 Generating Diagnostic Output

If the emulator supports generated detailed diagnostic output, the arguments are as follows.

Table 9 Emulator Diagnostic Output Arguments

Standard Extensions	Description
`-Xverbose`	Identical to `-Xverbose:all`.
`-Xverbose:`*type-list*	Display tracing output for the different types of information. *type-list* can be any combination of the following types. Multiple output types can be specified by separating them by a comma. For example, to see both Garbage Collection and Class Loading, use `-Xverbose:gc,classes`. Use `all` to see all output types. Write diagnostic output to the standard output of the emulator process. The format of the output is not specified.

	Type	Description
	`all`	All tracing options. **CAUTION:** Displaying all diagnostic information is very verbose and causes MIDlets to run slowly.
	`allocation`	Prints every allocation on the emulator's heap and displays overall heap usage statistics.
	`gc`	Prints every de-allocation on the emulator's heap, the inverse of `-Xverbose:allocation`.
	`gcverbose`	Print detailed analysis of the garbage collection process. **CAUTION:** This type produces lots of output.
	`class`	Print class loading, creation, and initialization.
	`classverbose`	Print detailed information as the different parts of a class file are loaded into the emulator. **CAUTION:** This type produces lots of output.
	`verifier`	Trace the emulator's internal class verifier.
	`stackmaps`	Print class stackmap information.
	`bytecodes`	Print each bytecode as the program executes.

		CAUTION: This type produces lots of output.
	`frames`	Print stack frame information when they are pushed and popped.
	`stackchunks`	Print stack and stack chunk creation.
	`exceptions`	Print all thrown exceptions, even if they are caught.
	`events`	Print events (such as PENDOWN) as they are received by the emulator.
	`threading`	Print status of all threads in the system.
	`monitors`	Print information whenever the program enters or exits a monitor. **CAUTION:** This type produces lots of output.
	`networking`	Print detailed information for every network related method. **CAUTION:** This type produces lots of output.

8.2 Connecting the Emulator to a Debugger

The emulator can support connecting an IDE to a debugger through the `-Xdebug` option. IDEs can test for this feature by using `-Xquery`.

Table 10 Emulator Debugger Arguments

`-Xdebug -Xrunjdwp :`*`name`*`=`*`value`*`[,`*`name`*`=`*`value`*`[...]]`		Enable runtime debugging. `-Xdebug` and `-Xrunjdwp` are always used together. The name and value pairs are used to control how a Java Debug Wire Protocol (JDWP) session is created. *name* is one of the options in the table below. Appropriate values are determined by the option name. Multiple name and value pairs can be separated by commas.
	Name	**Description**
	`transport`	The transport mechanism used to communicate with the debugger. The default value `dt_socket` is the only mechanism that must be supported.
	`address`	The transport address for the debugger connection. This can be either *host:port* or *port* formats. In the port-only format, the host is `localhost`.
	`server`	Start the debug agent as a server. The debugger must connect to the port specified. The value must be either `y` or `n`. The default is `n`.
	`suspend`	Suspend the VM immediately after establishing a connection with the debugger, or not. Values must be either `y` or `n`. The default is `y`.

8.3 Using the Emulator for Automated Testing

Another optional feature of the emulator is automated testing, exposed through the `-Xautotest` feature.

`emulator -Xautotest:`*URL*

Use `-Xautotest` to repeatedly download and run MIDlet suites from the specified URL. The URL must use the HTTP protocol. This feature is useful in conjunction with a server that returns a different test MIDlet each time the URL is accessed. You can think of `-Xautotest` as a repeating `-Xjam:transient` argument.

The emulator should continue to repeatedly download and run applications from the given URL until it receives an HTTP error when accessing the URL. The emulator should then exit.

9 Keytool Execution

If the `mekeytool` command is present, it must operate as described in this section.

`mekeytool` is used to add, remove, and list certificates. Certificates are used by an emulator for the following purposes:

- Verifying an application when it is installed by the application manager
- Establishing a secure connection using HTTPS or SSL

`mekeytool` must accept the arguments described in Table 11.

Table 11 `mekeytool` Arguments

Argument	Description
`-list`	Lists installed keys to standard output. The format of the output is defined below. After writing the list, `mekeytool` should terminate.
`-import` [`-keystore` *filename*] [`-storepass` *password*] `-alias` *key-alias* [`-domain` *domain*]	Import a key from an existing keystore.
`-delete -owner` *owner-name*	Delete the key with the given owner.
`-delete -number` *key-number*	Delete the key with the given number.

Provide only one of the arguments `-list`, `-import`, or `-delete`. Do not provide any arguments other than those listed in Table 11.

The format of the output from `mekeytool -list` is as follows:

Certificate-data
blank-line

Certificate-data consists of zero or more certificate blocks with the following form.

```
Key Certificate-number
  Owner: Owner-parameters
  Valid from Date to Date
  Security Domain: Domain-name
```

Certificate numbers start at 1 and go up 1 for every certificate.

Owner-parameters is a semicolon separated list of owner parameters. Each owner parameter looks like this:

key=value

Valid keys are `CN`, `OU`, `OR`, `LO`, `ST`, `CO` and `EM`.

A *Date* is in this form:

weekday month monthday time timezone year

Each field of this string has a fixed length.

weekday has three letters

month has three letters

monthday uses two digits to show the day of the month

time is expressed as two digits each for hours (on a 24 hour clock), minutes, and seconds, separated by colons, as in *HH*:*mm*:*ss*

timezone is three letters

year is four digits

The *Domain-name* for the certificate must be one of the values returned in the security.domains or *device-name*.security.domains properties returned from `-Xquery`.

The three lines following the `Key` line in each certificate block are indented two spaces.

For example:

```
C:\WTK23\bin>mekeytool -list
Key 1
  Owner: C=US;O=RSA Data Security, Inc.;OU=Secure Server Certification Authority
  Valid from Wed Nov 09 02:00:00 IST 1994 to Fri Jan 08 01:59:59 IST 2010
  Security Domain: untrusted
Key 2
  Owner: CN=Sun Microsystems Inc TEST CA;O=Sun Microsystems Inc
  Valid from Mon Nov 20 23:20:50 IST 2000 to Fri Nov 20 23:20:50 IST 2009
  Security Domain: trusted
Key 3
  Owner: CN=thehost;OU=JCT;O=dummy CA;L=Santa Clara;ST=CA;C=US
  Valid from Wed Jul 24 18:58:02 IDT 2002 to Sat Jul 21 18:58:02 IDT 2012
  Security Domain: trusted
```

10 API Manifests

JAR files containing emulator APIs in the `lib` directory should provide additional information to IDEs by providing manifest files with the attributes listed in Table 12.

Table 12 API Manifest Attributes

Attribute	Description
`API`	Code name for the API. This name is only used internally and has no significance beyond its use for calculating API dependencies.
`API-Name`	External name for the API. This text can be shown to developers.
`API-Specification-Version`	The version number of the specification that is implemented by this JAR file.
`API-Type`	Use one of the following values for this attribute. • `Configuration` - The API is a configuration such as CLDC. • `Profile` - The API is a profile such as MIDP or PDAP. • `Optional` - The API is an optional API such as MMAPI or WMA.
`API-Dependencies`	A comma-separated list of APIs that are required by the API contained in this JAR. Each API dependency can contain just the code name of the API, or can additionally contain `= version-number` to require that a specific version be installed, or `>= version-number` to specify that at least a certain version be installed.

For example, the manifest of one JAR file might contain the following:

```
API: CLDC
API-Name: Connected Limited Device Configuration
API-Specification-Version: 1.0
API-Type: Configuration
```

Another JAR file's manifest might contain the following:

```
API: MIDP
API-Name: Mobile Information Device Profile
API-Specification-Version: 1.0
API-Type: Profile
API-Dependencies: CLDC >= 1.0
```

An IDE can use API manifests in conjunction with the *device-name*`.apis` properties returned

from `-Xquery` to select the versions of APIs with which it builds an application.

For example, an emulator might report the following for the device `MyDevice`:

```
MyDevice.bootclasspath:
   C:/MyEmulator/lib/midp20.jar,C:/MyEmulator/lib/cldc11.jar
MyDevice.apis:
   C:/MyEmulator/lib/midp20.jar,C:/MyEmulator/lib/midp10.jar,C:/MyEmula
   tor/lib/cldc11.jar,C:/MyEmulator/lib/cldc10.jar
```

The `bootclasspath` property gives the default set of APIs, and the `apis` property includes earlier versions of APIs.

Examining the JAR files referred to in the `MyDevice.apis` property, an IDE finds the following data in the manifests.

Table 13 Example API Attributes

File	API	API-Name	API-Specification-Version	API-Type	API-Dependencies
`midp20.jar`	MIDP	Mobile Information Device Profile	2.0	Profile	CLDC
`midp10.jar`	MIDP	Mobile Information Device Profile	1.0	Profile	CLDC
`cldc11.jar`	CLDC	Connected Limited Device Configuration	1.1	Configuration	
`cldc10.jar`	CLDC	Connected Limited Device Configuration	1.0	Configuration	

From this data, the IDE concludes that the following combinations of APIs are valid for this device:

- CLDC 1.0, MIDP 1.0
- CLDC 1.0, MIDP 2.0
- CLDC 1.1, MIDP 1.0
- CLDC 1.1, MIDP 2.0

If the IDE then needs to build an application for CLDC 1.1 and MIDP 1.0, it uses the API files `cldc11.jar` and `midp10.jar`.