GNU Classpath Tools Guide

The GNU Classpath Team

Table of Contents

1 Applet Tools

Two Applet Tools are available with GNU Classpath: **appletviewer** and **gcjwebplugin**.

To avoid conflicts with other implementations, the appletviewer executable is called "gappletviewer".

If while using these tools you think you found a bug, then please report it at classpath-bugs.

1.1 The `appletviewer` Tool

SYNOPSIS

appletviewer [*OPTION*]... *URL*...

appletviewer [*OPTION*]... '`-code`' *CODE*

appletviewer [*OPTION*]... '`-plugin`' *INPUT*,*OUTPUT*

DESCRIPTION The `appletviewer` tool loads and runs an applet.

Use the first form to test applets specified by tag. The URL should resolve to an HTML document from which the `appletviewer` will extract applet tags. The APPLET, EMBED and OBJECT tags are supported. If a given document contains multiple applet tags, all the applets will be loaded, with each applet appearing in its own window. Likewise, when multiple URLs are specified, each applet tag instance is given its own window. If a given document contains no recognized tags the `appletviewer` does nothing.

```
appletviewer http://www.gnu.org/software/classpath/
```

Use the second form to test an applet in development. This form allows applet tag attributes to be supplied on the command line. Only one applet may be specified using the '`-code`' option. The '`-code`' option overrides the URL form – any URLs specified will be ignored.

```
appletviewer -code Test.class -param datafile,data.txt
```

`gcjwebplugin` uses the third form to communicate with the `appletviewer` through named pipes.

URL OPTIONS

`-debug` This option is not yet implemented but is provided for compatibility.

`-encoding` *CHARSET*
: Use this option to specify an alternate character encoding for the specified HTML page.

APPLET TAG OPTIONS

`-code` *CODE*
: Use the '`-code`' option to specify the value of the applet tag *CODE* attribute.

`-codebase` *CODEBASE*
: Use the '`-codebase`' option to specify the value of the applet tag *CODEBASE* attribute.

`-archive` *ARCHIVE*
: Use the '`-archive`' option to specify the value of the applet tag *ARCHIVE* attribute.

`-width` *WIDTH*
: Use the '`-width`' option to specify the value of the applet tag *WIDTH* attribute.

`-height` *HEIGHT*
: Use the '`-height`' option to specify the value of the applet tag *HEIGHT* attribute.

`-param` *NAME*`,`*VALUE*
: Use the '`-param`' option to specify values for the *NAME* and *VALUE* attributes of an applet PARAM tag.

PLUGIN OPTION

`-plugin` *INPUT*`,`*OUTPUT*
: `gcjwebplugin` uses the '`-plugin`' option to specify the named pipe the `appletviewer` should use for receiving commands (*INPUT*) and the one it should use for sending commands to `gcjwebplugin` (*OUTPUT*).

DEBUGGING OPTION

`-verbose` Use the '`-verbose`' option to have the `appletviewer` print debugging messages.

STANDARD OPTIONS

`-help` Use the '`-help`' option to have the `appletviewer` print a usage message, then exit.

`-version` Use the '`-version`' option to have the `appletviewer` print its version, then exit.

`-J`*OPTION* Use the '`-J`' option to pass *OPTION* to the virtual machine that will run the `appletviewer`. Unlike other options, there must not be a space between the '`-J`' and *OPTION*.

1.2 The `gcjwebplugin` Tool

`gcjwebplugin` is a plugin that adds applet support to web browsers. Currently `gcjwebplugin` only supports Mozilla-based browsers (e.g., Firefox, Galeon, Mozilla).

2 Security Tools

Two Security Tools are available with GNU Classpath: `jarsigner` and `keytool`.

To avoid conflicts with other implementations, the jarsigner executable is called `gjarsigner` and the keytool executable is called `gkeytool`.

If while using these tools you think you found a bug, then please report it at classpath-bugs.

2.1 The `jarsigner` Tool

The `jarsigner` tool is invoked from the command line, in one of two forms, as follows:

```
jarsigner [OPTION]... FILE ALIAS

jarsigner '-verify' [OPTION]... FILE
```

When the first form is used, the tool signs the designated JAR file. The second form, on the other hand, is used to verify a previously signed JAR file.

FILE is the .JAR file to process; i.e., to sign if the first syntax form is used, or to verify if the second syntax form is used instead.

ALIAS must be a known *Alias* of a *Key Entry* in the designated *Key Store*. The private key material associated with this *Alias* is then used for signing the designated .JAR file.

2.1.1 Common options

The following options may be used when the tool is used for either signing, or verifying, a .JAR file.

`-verbose` Use this option to force the tool to generate more verbose messages, during its processing.

`-internalsf`
When present, the tool will include –which otherwise it does not– the `.SF` file in the `.DSA` generated file.

`-sectionsonly`
When present, the tool will include in the `.SF` generated file –which otherwise it does not– a header containing a hash of the whole manifest file. When that header is included, the tool can quickly check, during verification, if the hash (in the header) matches or not the manifest file.

`-provider PROVIDER_CLASS_NAME`
A fully qualified class name of a *Security Provider* to add to the current list of *Security Providers* already installed in the JVM in-use. If a provider class is specified with this option, and was successfully added to the runtime –i.e. it was not already installed– then the tool will attempt to remove this *Security Provider* before exiting.

`-help` Prints a help text similar to this one.

2.1.2 Signing options

The following options may be specified when using the tool for signing purposes.

`-keystore` *`URL`*

Use this option to specify the location of the key store to use. The default value is a file URL referencing the file named '`.keystore`' located in the path returned by the call to `java.lang.System#getProperty(String)` using `user.home` as argument.

If a URL was specified, but was found to be malformed –e.g. missing protocol element– the tool will attempt to use the URL value as a file-name (with absolute or relative path-name) of a key store –as if the protocol was `file:`.

`-storetype` *`STORE_TYPE`*

Use this option to specify the type of the key store to use. The default value, if this option is omitted, is that of the property `keystore.type` in the security properties file, which is obtained by invoking the static method call `getDefaultType()` in `java.security.KeyStore`.

`-storepass` *`PASSWORD`*

Use this option to specify the password which will be used to unlock the key store. If this option is missing, the User will be prompted to provide a password.

`-keypass` *`PASSWORD`*

Use this option to specify the password which the tool will use to unlock the *Key Entry* associated with the designated *Alias*.

If this option is omitted, the tool will first attempt to unlock the *Key Entry* using the same password protecting the key store. If this fails, you will then be prompted to provide a password.

`-sigfile` *`NAME`*

Use this option to designate a literal that will be used to construct file names for both the `.SF` and `.DSA` signature files. These files will be generated, by the tool, and placed in the '`META-INF`' directory of the signed JAR. Permissible characters for *NAME* must be in the range `"a-zA-Z0-9_-"`. All characters will be converted to upper-case ones.

If this option is missing, the first eight characters of the *ALIAS* argument will be used. When this is the case, any character in *ALIAS* that is outside the permissible range of characters will be replaced by an underscore.

`-signedjar` *`FILE`*

Use this option to specify the file name of the signed JAR. If this option is omitted, then the signed JAR will be named the same as *FILE*; i.e., the input JAR file will be replaced with the signed copy.

2.1.3 Verification options

The following options may be specified when using the tool for verification purposes.

`-verify` Use this option to indicate that the tool is to be used for verification purposes.

`-certs` This option is used in conjunction with the '`-verbose`' option. When present, along with the '`-verbose`' option, the tool will print more detailed information about the certificates of the signer(s) being processed.

2.2 The `keytool` Tool

Cryptographic credentials, in a Java environment, are usually stored in a *Key Store*. The Java SDK specifies a *Key Store* as a persistent container of two types of objects: *Key Entries* and *Trusted Certificates*. The security tool `keytool` is a Java-based application for managing those types of objects.

A *Key Entry* represents the private key part of a key-pair used in Public-Key Cryptography, and a signed X.509 certificate which authenticates the public key part for a known entity; i.e. the owner of the key-pair. The X.509 certificate itself contains the public key part of the key-pair.

A *Trusted Certificate* is a signed X.509 certificate issued by a trusted entity. The *Trust* in this context is relative to the User of the `keytool`. In other words, the existence of a *Trusted Certificate* in the *Key Store* processed by a `keytool` command implies that the User trusts the *Issuer* of that *Trusted Certificate* to also sign, and hence authenticates, other *Subjects* the tool may process.

Trusted Certificates are important because they allow the tool to mechanically construct *Chains of Trust* starting from one of the *Trusted Certificates* in a *Key Store* and ending with a certificate whose *Issuer* is potentially unknown. A valid chain is an ordered list, starting with a *Trusted Certificate* (also called the *anchor*), ending with the target certificate, and satisfying the condition that the *Subject* of certificate `#i` is the *Issuer* of certificate `#i + 1`.

The `keytool` is invoked from the command line as follows:

```
keytool [COMMAND] ...
```

Multiple *COMMAND*s may be specified at once, each complete with its own options. `keytool` will parse all the arguments, before processing, and executing, each `COMMAND`. If an exception occurs while executing one *COMMAND* `keytool` will abort. Note however that because the implementation of the tool uses code to parse command line options that also supports GNU-style options, you have to separate each command group with a double-hyphen; e.g

```
keytool -list -- -printcert -alias mykey
```

Here is a summary of the commands supported by the tool:

1. Add/Update commands

   `-genkey [`*`OPTION`*`]...`
   Generate a new *Key Entry*, eventually creating a new key store.

   `-import [`*`OPTION`*`]...`
   Add, to a key store, *Key Entries* (private keys and certificate chains authenticating the public keys) and *Trusted Certificates* (3rd party certificates which can be used as *Trust Anchors* when building chains-of-trust).

   `-selfcert [`*`OPTION`*`]...`
   Generate a new self-signed *Trusted Certificate*.

`-cacert [OPTION]...`
: Import a CA *Trusted Certificate*.

`-identitydb [OPTION]...`
: **NOT IMPLEMENTED YET**.
Import a JDK 1.1 style Identity Database.

2. Export commands

`-certreq [OPTION]...`
: Issue a *Certificate Signing Request* (CSR) which can be then sent to a *Certification Authority* (CA) to issue a certificate signed (by the CA) and authenticating the *Subject* of the request.

`-export [OPTION]...`
: Export a certificate from a key store.

3. Display commands

`-list [OPTION]...`
: Print one or all certificates in a key store to `STDOUT`.

`-printcert [OPTION]...`
: Print a human-readable form of a certificate, in a designated file, to `STDOUT`.

4. Management commands

`-keyclone [OPTION]...`
: Clone a *Key Entry* in a key store.

`-storepasswd [OPTION]...`
: Change the password protecting a key store.

`-keypasswd [OPTION]...`
: Change the password protecting a *Key Entry* in a key store.

`-delete [OPTION]...`
: Delete a *Key Entry* or a *Trusted Certificate* from a key store.

2.2.1 Getting help

To get a general help text about the tool, use the `-help` option; e.g.

```
keytool -help
```

To get more specific help text about one of the tool's command use the `-help` option for that command; e.g.

```
keytool -genkey -help
```

In both instances, the tool will print a help text and then will exit the running JVM.

It is worth noting here that the help messages printed by the tool are I18N-ready. This means that if/when the contents of the tool's *Message Bundle* properties file are available in languages other than English, you may see those messages in that language.

2.2.2 Common options

The following '`OPTION`'s are used in more than one `COMMAND`. They are described here to reduce redundancy.

`-alias` *`Alias`*

Every entry, be it a *Key Entry* or a *Trusted Certificate*, in a key store is uniquely identified by a user-defined *Alias* string. Use this option to specify the *Alias* to use when referring to an entry in the key store. Unless specified otherwise, a default value of `mykey` shall be used when this option is omitted from the command line.

`-keyalg` *`ALGORITHM`*

Use this option to specify the canonical name of the key-pair generation algorithm. The default value for this option is `DSS` (a synonym for the Digital Signature Algorithm also known as DSA).

`-keysize` *`SIZE`*

Use this option to specify the number of bits of the shared modulus (for both the public and private keys) to use when generating new keys. A default value of `1024` will be used if this option is omitted from the command line.

`-validity` *`DAY_COUNT`*

Use this option to specify the number of days a newly generated certificate will be valid for. The default value is `90` (days) if this option is omitted from the command line.

`-storetype` *`STORE_TYPE`*

Use this option to specify the type of the key store to use. The default value, if this option is omitted, is that of the property `keystore.type` in the security properties file, which is obtained by invoking the static method call `getDefaultType()` in `java.security.KeyStore`.

`-storepass` *`PASSWORD`*

Use this option to specify the password protecting the key store. If this option is omitted from the command line, you will be prompted to provide a password.

`-keystore` *`URL`*

Use this option to specify the location of the key store to use. The default value is a file URL referencing the file named '`.keystore`' located in the path returned by the call to `java.lang.System#getProperty(String)` using `user.home` as argument.

If a URL was specified, but was found to be malformed –e.g. missing protocol element– the tool will attempt to use the URL value as a file-name (with absolute or relative path-name) of a key store –as if the protocol was `file:`.

`-provider` *`PROVIDER_CLASS_NAME`*

A fully qualified class name of a *Security Provider* to add to the current list of *Security Providers* already installed in the JVM in-use. If a provider class is specified with this option, and was successfully added to the runtime –i.e. it was not already installed– then the tool will attempt to removed this *Security Provider* before exiting.

`-file` *FILE*

Use this option to designate a file to use with a command. When specified with this option, the value is expected to be the fully qualified path of a file accessible by the File System. Depending on the command, the file may be used as input or as output. When this option is omitted from the command line, `STDIN` will be used instead, as the source of input, and `STDOUT` will be used instead as the output destination.

`-v` Unless specified otherwise, use this option to enable more verbose output.

2.2.3 X.500 Distinguished Names

A *Distinguished Name* (or DN) MUST be supplied with some of the `COMMAND`s using a `-dname` option. The syntax of a valid value for this option MUST follow RFC-2253 specifications. Namely the following components (with their accepted meaning) will be recognized. Note that the component name is case-insensitive:

CN The Common Name; e.g. *`host.domain.com`*

OU The Organizational Unit; e.g. *`IT Department`*

O The Organization Name; e.g. *`The Sample Company`*

L The Locality Name; e.g. *`Sydney`*

ST The State Name; e.g. *`New South Wales`*

C The 2-letter Country identifier; e.g. *`AU`*

When specified with a `-dname` option, each pair of component/value will be separated from the other with a comma. Each component and value pair MUST be separated by an equal sign. For example, the following is a valid DN value:

CN=host.domain.com, O=The Sample Company, L=Sydney, ST=NSW, C=AU

If the *Distinguished Name* is required, and no valid default value can be used, the tool will prompt you to enter the information through the console.

2.2.4 Add/Update commands

2.2.4.1 The '`-genkey`' command

Use this command to generate a new key-pair (both private and public keys), and save these credentials in the key store as a *Key Entry*, associated with the designated (if was specified with the '`-alias`' option) or default (if the '`-alias`' option is omitted) *Alias*.

The private key material will be protected with a user-defined password (see '`-keypass`' option). The public key on the other hand will be part of a self-signed X.509 certificate, which will form a 1-element chain and will be saved in the key store.

`-alias` *ALIAS*

For more details see [ALIAS], page 7.

`-keyalg` *ALGORITHM*

For more details see [ALGORITHM], page 7.

`-keysize` *KEY_SIZE*
: For more details see [KEY_SIZE], page 7.

`-sigalg` *ALGORITHM*
: The canonical name of the digital signature algorithm to use for signing certificates. If this option is omitted, a default value will be chosen based on the type of the key-pair; i.e., the algorithm that ends up being used by the -keyalg option. If the key-pair generation algorithm is `DSA`, the value for the signature algorithm will be `SHA1withDSA`. If on the other hand the key-pair generation algorithm is `RSA`, then the tool will use `MD5withRSA` as the signature algorithm.

`-dname` *NAME*
: This a mandatory value for the command. If no value is specified –i.e. the '`-dname`' option is omitted– the tool will prompt you to enter a *Distinguished Name* to use as both the *Owner* and *Issuer* of the generated self-signed certificate.

  For more details see [X.500 DISTINGUISHED NAME], page 8.

`-keypass` *PASSWORD*
: Use this option to specify the password which the tool will use to protect the newly created *Key Entry*.

  If this option is omitted, you will be prompted to provide a password.

`-validity` *DAY_COUNT*
: For more details see [DAY_COUNT], page 7.

`-storetype` *STORE_TYPE*
: For more details see [STORE_TYPE], page 7.

`-keystore` *URL*
: For more details see [URL], page 7.

`-storepass` *PASSWORD*
: For more details see [PASSWORD], page 7.

`-provider` *PROVIDER_CLASS_NAME*
: For more details see [PROVIDER_CLASS_NAME], page 7.

`-v`
: For more details see [verbose], page 8.

2.2.4.2 The '`-import`' command

Use this command to read an X.509 certificate, or a PKCS#7 *Certificate Reply* from a designated input source and incorporate the certificates into the key store.

If the *Alias* does not already exist in the key store, the tool treats the certificate read from the input source as a new *Trusted Certificate*. It then attempts to discover a chain-of-trust, starting from that certificate and ending at another *Trusted Certificate*, already stored in the key store. If the '`-trustcacerts`' option is present, an additional key store, of type `JKS` named '`cacerts`', and assumed to be present in '`${JAVA_HOME}/lib/security`' will also be consulted if found –`${JAVA_HOME}` refers to the location of an installed *Java Runtime Environment* (JRE). If no chain-of-trust can be established, and unless the `-noprompt` option has been specified, the certificate is printed to `STDOUT` and the user is prompted for a confirmation.

If *Alias* exists in the key store, the tool will treat the certificate(s) read from the input source as a *Certificate Reply*, which can be a chain of certificates, that eventually would replace the chain of certificates associated with the *Key Entry* of that *Alias*. The substitution of the certificates only occurs if a chain-of-trust can be established between the bottom certificate of the chain read from the input file and the *Trusted Certificates* already present in the key store. Again, if the '`-trustcacerts`' option is specified, additional *Trusted Certificates* in the same '`cacerts`' key store will be considered. If no chain-of-trust can be established, the operation will abort.

`-alias` *`ALIAS`*
: For more details see [ALIAS], page 7.

`-file` *`FILE`*
: For more details see [FILE], page 7.

`-keypass` *`PASSWORD`*
: Use this option to specify the password which the tool will use to protect the *Key Entry* associated with the designated *Alias*, when replacing this *Alias*' chain of certificates with that found in the certificate reply.

  If this option is omitted, and the chain-of-trust for the certificate reply has been established, the tool will first attempt to unlock the *Key Entry* using the same password protecting the key store. If this fails, you will then be prompted to provide a password.

`-noprompt`
: Use this option to prevent the tool from prompting the user.

`-trustcacerts`
: Use this option to indicate to the tool that a key store, of type `JKS`, named '`cacerts`', and usually located in '`lib/security`' in an installed *Java Runtime Environment* should be considered when trying to establish chain-of-trusts.

`-storetype` *`STORE_TYPE`*
: For more details see [STORE_TYPE], page 7.

`-keystore` *`URL`*
: For more details see [URL], page 7.

`-storepass` *`PASSWORD`*
: For more details see [PASSWORD], page 7.

`-provider` *`PROVIDER_CLASS_NAME`*
: For more details see [PROVIDER_CLASS_NAME], page 7.

`-v` For more details see [verbose], page 8.

2.2.4.3 The '`-selfcert`' command

Use this command to generate a self-signed X.509 version 1 certificate. The newly generated certificate will form a chain of one element which will replace the previous chain associated with the designated *Alias* (if '`-alias`' option was specified), or the default *Alias* (if '`-alias`' option was omitted).

`-alias` *`ALIAS`*
: For more details see [ALIAS], page 7.

`-sigalg` *`ALGORITHM`*
: The canonical name of the digital signature algorithm to use for signing the certificate. If this option is omitted, a default value will be chosen based on the type of the private key associated with the designated *Alias*. If the private key is a `DSA` one, the value for the signature algorithm will be `SHA1withDSA`. If on the other hand the private key is an `RSA` one, then the tool will use `MD5withRSA` as the signature algorithm.

`-dname` *`NAME`*
: Use this option to specify the *Distinguished Name* of the newly generated self-signed certificate. If this option is omitted, the existing *Distinguished Name* of the base certificate in the chain associated with the designated *Alias* will be used instead.

  For more details see [X.500 DISTINGUISHED NAME], page 8.

`-validity` *`DAY_COUNT`*
: For more details see [DAY_COUNT], page 7.

`-keypass` *`PASSWORD`*
: Use this option to specify the password which the tool will use to unlock the *Key Entry* associated with the designated *Alias*.

  If this option is omitted, the tool will first attempt to unlock the *Key Entry* using the same password protecting the key store. If this fails, you will then be prompted to provide a password.

`-storetype` *`STORE_TYPE`*
: For more details see [STORE_TYPE], page 7.

`-keystore` *`URL`*
: For more details see [URL], page 7.

`-storepass` *`PASSWORD`*
: For more details see [PASSWORD], page 7.

`-provider` *`PROVIDER_CLASS_NAME`*
: For more details see [PROVIDER_CLASS_NAME], page 7.

`-v`
: For more details see [verbose], page 8.

2.2.4.4 The '`-cacert`' command

Use this command to import, a CA certificate and add it to the key store as a *Trusted Certificate*. The *Alias* for this new entry will be constructed from the FILE's base-name after replacing hyphens and dots with underscores.

This command is useful when used in a script that recursively visits a directory of CA certificates to populate a `cacerts.gkr` *Key Store* of trusted certificates which can then be used commands that specify the '`-trustcacerts`' option.

`-file` *`FILE`*
: For more details see [FILE], page 7.

`-storetype` *STORE_TYPE*
: For more details see [STORE_TYPE], page 7.

`-keystore` *URL*
: For more details see [URL], page 7.

`-storepass` *PASSWORD*
: For more details see [PASSWORD], page 7.

`-provider` *PROVIDER_CLASS_NAME*
: For more details see [PROVIDER_CLASS_NAME], page 7.

`-v`
: For more details see [verbose], page 8.

2.2.4.5 The '`-identitydb`' command

NOT IMPLEMENTED YET.

Use this command to import a JDK 1.1 style Identity Database.

`-file` *FILE*
: For more details see [FILE], page 7.

`-storetype` *STORE_TYPE*
: For more details see [STORE_TYPE], page 7.

`-keystore` *URL*
: For more details see [URL], page 7.

`-storepass` *PASSWORD*
: For more details see [PASSWORD], page 7.

`-provider` *PROVIDER_CLASS_NAME*
: For more details see [PROVIDER_CLASS_NAME], page 7.

`-v`
: For more details see [verbose], page 8.

2.2.5 Export commands

2.2.5.1 The '`-certreq`' command

Use this command to generate a PKCS#10 *Certificate Signing Request* (CSR) and write it to a designated output destination. The contents of the destination should look something like the following:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MI...QAwXzEUMBIGA1UEAwwLcnNuQGdudS5vcmcxGzAZBgNVBAoMElUg
Q2...A0GA1UEBwwGU3lkbmV5MQwwCgYDVQQIDANOU1cxCzAJBgNVBACC
...
FC...IVwNVOfQLRX+O5kAhQ/a4RTZme2L8PnpvgRwrf7Eg8D6w==
-----END NEW CERTIFICATE REQUEST-----
```

IMPORTANT: Some documentation (e.g. RSA examples) claims that the `Attributes` field, in the CSR is `OPTIONAL` while RFC-2986 implies the opposite. This implementation considers this field, by default, as `OPTIONAL`, unless the option '`-attributes`' is specified on the command line.

`-alias` *ALIAS*
: For more details see [ALIAS], page 7.

`-sigalg` *ALGORITHM*
: The canonical name of the digital signature algorithm to use for signing the certificate. If this option is omitted, a default value will be chosen based on the type of the private key associated with the designated *Alias*. If the private key is a `DSA` one, the value for the signature algorithm will be `SHA1withDSA`. If on the other hand the private key is an `RSA` one, then the tool will use `MD5withRSA` as the signature algorithm.

`-file` *FILE*
: For more details see [FILE], page 7.

`-keypass` *PASSWORD*
: Use this option to specify the password which the tool will use to unlock the *Key Entry* associated with the designated *Alias*.

  If this option is omitted, the tool will first attempt to unlock the *Key Entry* using the same password protecting the key store. If this fails, you will then be prompted to provide a password.

`-storetype` *STORE_TYPE*
: For more details see [STORE_TYPE], page 7.

`-keystore` *URL*
: For more details see [URL], page 7.

`-storepass` *PASSWORD*
: For more details see [PASSWORD], page 7.

`-provider` *PROVIDER_CLASS_NAME*
: For more details see [PROVIDER_CLASS_NAME], page 7.

`-v`
: For more details see [verbose], page 8.

`-attributes`
: Use this option to force the tool to encode a `NULL` DER value in the CSR as the value of the `Attributes` field.

2.2.5.2 The '`-export`' command

Use this command to export a certificate stored in a key store to a designated output destination, either in binary format (if the '`-v`' option is specified), or in RFC-1421 compliant encoding (if the '`-rfc`' option is specified instead).

`-alias` *ALIAS*
: For more details see [ALIAS], page 7.

`-file` *FILE*
: For more details see [FILE], page 7.

`-storetype` *STORE_TYPE*
: For more details see [STORE_TYPE], page 7.

`-keystore` *URL*
: For more details see [URL], page 7.

`-storepass` *PASSWORD*
: For more details see [PASSWORD], page 7.

`-provider` *PROVIDER_CLASS_NAME*
: For more details see [PROVIDER_CLASS_NAME], page 7.

`-rfc`
: Use RFC-1421 specifications when encoding the output.

`-v`
: Output the certificate in binary DER encoding. This is the default output format of the command if neither '`-rfc`' nor `-v` options were detected on the command line. If both this option and the '`-rfc`' option are detected on the command line, the tool will opt for the RFC-1421 style encoding.

2.2.6 Display commands

2.2.6.1 The '`-list`' command

Use this command to print one or all of a key store entries to `STDOUT`. Usually this command will only print a *fingerprint* of the certificate, unless either the '`-rfc`' or the '`-v`' option is specified.

`-alias` *ALIAS*
: If this option is omitted, the tool will print ALL the entries found in the key store.

  For more details see [ALIAS], page 7.

`-storetype` *STORE_TYPE*
: For more details see [STORE_TYPE], page 7.

`-keystore` *URL*
: For more details see [URL], page 7.

`-storepass` *PASSWORD*
: For more details see [PASSWORD], page 7.

`-provider` *PROVIDER_CLASS_NAME*
: For more details see [PROVIDER_CLASS_NAME], page 7.

`-rfc`
: Use RFC-1421 specifications when encoding the output.

`-v`
: Output the certificate in human-readable format. If both this option and the '`-rfc`' option are detected on the command line, the tool will opt for the human-readable form and will not abort the command.

2.2.6.2 The '`-printcert`' command

Use this command to read a certificate from a designated input source and print it to `STDOUT` in a human-readable form.

`-file` *FILE*
: For more details see [FILE], page 7.

`-v`
: For more details see [verbose], page 8.

2.2.7 Management commands

2.2.7.1 The '-keyclone' command

Use this command to clone an existing *Key Entry* and store it under a new (different) *Alias* protecting, its private key material with possibly a new password.

`-alias` *ALIAS*
: For more details see [ALIAS], page 7.

`-dest` *ALIAS*
: Use this option to specify the new *Alias* which will be used to identify the cloned copy of the *Key Entry*.

`-keypass` *PASSWORD*
: Use this option to specify the password which the tool will use to unlock the *Key Entry* associated with the designated *Alias*.

  If this option is omitted, the tool will first attempt to unlock the *Key Entry* using the same password protecting the key store. If this fails, you will then be prompted to provide a password.

`-new` *PASSWORD*
: Use this option to specify the password protecting the private key material of the newly cloned copy of the *Key Entry*.

`-storetype` *STORE_TYPE*
: For more details see [STORE_TYPE], page 7.

`-keystore` *URL*
: For more details see [URL], page 7.

`-storepass` *PASSWORD*
: For more details see [PASSWORD], page 7.

`-provider` *PROVIDER_CLASS_NAME*
: For more details see [PROVIDER_CLASS_NAME], page 7.

`-v`
: For more details see [verbose], page 8.

2.2.7.2 The '-storepasswd' command

Use this command to change the password protecting a key store.

`-new` *PASSWORD*
: The new, and different, password which will be used to protect the designated key store.

`-storetype` *STORE_TYPE*
: For more details see [STORE_TYPE], page 7.

`-keystore` *URL*
: For more details see [URL], page 7.

`-storepass` *PASSWORD*
: For more details see [PASSWORD], page 7.

`-provider` *PROVIDER_CLASS_NAME*
: For more details see [PROVIDER_CLASS_NAME], page 7.

`-v`
: For more details see [verbose], page 8.

2.2.7.3 The '`-keypasswd`' command

Use this command to change the password protecting the private key material of a designated *Key Entry*.

`-alias` *ALIAS*
: For more details see [ALIAS], page 7.

  Use this option to specify the password which the tool will use to unlock the *Key Entry* associated with the designated *Alias*.

  If this option is omitted, the tool will first attempt to unlock the *Key Entry* using the same password protecting the key store. If this fails, you will then be prompted to provide a password.

`-new` *PASSWORD*
: The new, and different, password which will be used to protect the private key material of the designated *Key Entry*.

`-storetype` *STORE_TYPE*
: For more details see [STORE_TYPE], page 7.

`-keystore` *URL*
: For more details see [URL], page 7.

`-storepass` *PASSWORD*
: For more details see [PASSWORD], page 7.

`-provider` *PROVIDER_CLASS_NAME*
: For more details see [PROVIDER_CLASS_NAME], page 7.

`-v`
: For more details see [verbose], page 8.

2.2.7.4 The '`-delete`' command

Use this command to delete a designated key store entry.

`-alias` *ALIAS*
: For more details see [ALIAS], page 7.

`-storetype` *STORE_TYPE*
: For more details see [STORE_TYPE], page 7.

`-keystore` *URL*
: For more details see [URL], page 7.

`-storepass` *PASSWORD*
: For more details see [PASSWORD], page 7.

`-provider` *PROVIDER_CLASS_NAME*
: For more details see [PROVIDER_CLASS_NAME], page 7.

`-v`
: For more details see [verbose], page 8.

3 Other Tools

This is a list of currently undocumented classpath tools: **jar**, **javah**, **gcjh**, **native2ascii**, **orbd**, **serialver**, **rmid**, **rmiregistry** and **tnameserv**.

3.1 The `jar` Tool

`gjar` is an implementation of Sun's jar utility that comes with the JDK.

If any file is a directory then it is processed recursively. The manifest file name and the archive file name needs to be specified in the same order the '`-m`' and '`-f`' flags are specified.

Operation mode:

`-c` Create new archive.

`-t` List table of contents for archive.

`-x` Extract named (or all) files from archive.

`-u` Update existing archive.

`-i` *`FILE`* Compute archive index.

Operation modifiers:

`-f` *`FILE`* Specify archive file name.

`-0` Store only; use no ZIP compression.

`-v` Generate verbose output on standard output.

`-M` Do not create a manifest file for the entries.

`-m` *`manifest`*
Include manifest information from specified *manifest* file.

File name selection:

`-C` *`DIR FILE`*
Change to the *DIR* and include the following *FILE*.

`-@` Read the names of the files to add to the archive from stdin. This option is supported only in combination with '`-c`' or '`-u`'. Non standard option added in the GCC version.

Standard options:

`-help` Print help text, then exit.

`-version` Print version number, then exit.

`-`*`JOPTION`* Pass argument to the Java runtime.

java(1), . . .

3.2 The `javah` Tool

The `gjavah` program is used to generate header files from class files. It can generate both CNI and JNI header files, as well as stub implementation files which can be used as a basis for implementing the required native methods.

`-d` *DIR* Set output directory.

`-o` *FILE* Set output file (only one of '`-d`' or '`-o`' may be used).

`-cmdfile` *FILE*
Read command file.

`-all` *DIR* Operate on all class files under directory *DIR*.

`-stubs` Emit stub implementation.

`-jni` Emit JNI stubs or header (default).

`-cni` Emit CNI stubs or header (default JNI).

`-verbose` Set verbose mode.

`-force` Output files should always be written.

Class path options:

`-classpath` *PATH*
Set the class path.

`-I`*DIR* Add directory to class path.

`-bootclasspath` *PATH*
Set the boot class path.

`-extdirs` *PATH*
Set the extension directory path.

Standard options:

`-help` Print help text, then exit.

`-version` Print version number, then exit.

`-`*JOPTION* Pass argument to the Java runtime.

javac(1), . . .

3.3 The `gcjh` Tool

The `gcjh` program is used to generate header files from class files. It can generate both CNI and JNI header files, as well as stub implementation files which can be used as a basis for implementing the required native methods. It is similar to `javah` but has slightly different command line options, and defaults to CNI.

See `javah` for a full description; this page only lists the additional options provided by `gcjh`.

CNI text options

`-add` *text*
: Insert *text* into class body.

`-append` *text*
: Append *text* after class declaration.

`-friend` *text*
: Insert *text* as a `friend` declaration.

`-prepend` *text*
: Insert *text* before start of class.

Compatibility options (unused)

`-td` *DIR*
`-M`
`-MM`
`-MD`
`-MMD`
: Unused compatibility option.

Standard options:

`-help`
: Print help text, then exit.

`-version`
: Print version number, then exit.

`-`*JOPTION*
: Pass argument to the Java runtime.

javac(1), javah(1), . . .

3.4 The `native2ascii` Tool

To be written . . .

`-encoding` *NAME*
: Set the encoding to use.

`-reversed`
: Convert from encoding to native.

Standard options:

`-help`
: Print help text, then exit.

`-version`
: Print version number, then exit.

`-`*JOPTION*
: Pass argument to the Java runtime.

javac(1), . . .

3.5 The `orbd` object request broker daemon

To be written . . .

`-ORBInitialPort` *PORT*
: Port on which persistent naming service is to be started.

`-ior` *FILE*
: File in which to store persistent naming service's IOR reference

`-directory` *DIR*
: Directory in which to store persistent data.

`-restart` Restart persistent naming service, clearing persistent naming database.

Standard options:

`-help` Print help text, then exit.

`-version` Print version number, then exit.

`-`*JOPTION* Pass argument to the Java runtime.

java(1), . . .

3.6 The `serialver` version command

Print the serialVersionUID of the specified classes.

`-classpath` *PATH*
: Class path to use to find classes.

Standard options:

`-help` Print help text, then exit.

`-version` Print version number, then exit.

`-`*JOPTION* Pass argument to the Java runtime.

javac(1), . . .

3.7 The `rmid` RMI activation system daemon

`rmiregistry` starts a remote object registry on the current host. If no port number is specified, then port 1099 is used.

Activation process control:

`-port` *PORT*
: Port on which activation system is to be started.

`-restart` Restart activation system, clearing persistent naming database, if any.

`-stop` Stop activation system.

Persistence:

`-persistent`
: Make activation system persistent.

`-directory` *DIR*
: Directory in which to store persistent data.

Debugging:

`-verbose` Log binding events to standard out.

Standard options:

`-help` Print help text, then exit.

`-version` Print version number, then exit.

`-JOPTION` Pass argument to the Java runtime.

java(1), . . .

3.8 The `rmiregistry` Tool

`grmiregistry` starts a remote object registry on the current host. If no port number is specified, then port 1099 is used.

Registry process control:

`-restart` Restart RMI naming service, clearing persistent naming database, if any.

`-stop` Stop RMI naming service.

Persistence:

`-persistent`
Make RMI naming service persistent.

`-directory` *DIR*
Directory in which to store persistent data.

Debugging:

`-verbose` Log binding events to standard out.

Standard options:

`-help` Print help text, then exit.

`-version` Print version number, then exit.

`-JOPTION` Pass argument to the Java runtime.

java(1), . . .

3.9 The `tnameserv` Tool

To be written . . .

`-ORBInitialPort` *PORT*
Port on which naming service is to be started.

`-ior` *FILE*
File in which to store naming service's IOR reference.

Standard options:

`-help` Print help text, then exit.

`-version` Print version number, then exit.

`-JOPTION` Pass argument to the Java runtime.

java(1), . . .

Info entry for '`gjdoc`'. Please report bugs to `http://savannah.gnu.org/bugs/?group=classpath`.
Julian Scheid

4 Generating HTML Documentation

Gjdoc can be used in two ways: as a stand-alone documentation tool, or as a driver for a user-specified Doclet. See Chapter 5 [Other Doclets], page 34.

In the default mode, Gjdoc will use the Standard Doclet '`HtmlDoclet`' to generate a set of HTML pages. The canonical usage is:

```
gjdoc -s src/java/ -all -d api-docs/
```

Here, '`src/java/`' is the root of your source code class hierarchy, '`-all`' means that all valid Java files found under this root directory should be processed, and '`api-docs/`' is the directory where the generated documentation should be placed.

To learn more about running Doclets other than the Standard Doclet, refer to the manual. See Section 4.13 [Invoking a Custom Doclet], page 33.

4.1 Invoking the Standard Doclet

Running the Gjdoc Standard Doclet '`HtmlDoclet`' is the default mode of operation for Gjdoc. This section lists the command line options you can specify in this mode. It doesn't distinguish between general Gjdoc options and options specific to the Standard Doclet.

If you want to learn which options are accepted when Gjdoc is used as a doclet driver, See Section 4.13 [Invoking a Custom Doclet], page 33.

4.2 Option Summary by Type

Here is a summary of all the options of both Gjdoc and the Standard Doclet, grouped by type. Explanations are in the following sections.

Source Set Options
: See Section 4.3 [Options For Specifying the Source Files To Operate on], page 23.

```
-sourcepath pathlist -subpackages pkglist -exclude pkglist
```

Source Format Options
: See Section 4.4 [Options For Specifying the Source Format], page 24.

```
-source release -encoding encoding -breakiterator
```

Interlinking Options
: See Section 4.5 [Options For Specifying the Source Files To Operate on], page 24.

```
-link url -linkoffline url file -noqualifier pkg:pkg:...
```

Generation Options
: See Section 4.6 [Options Controlling What is Included in the Output], page 25.

```
-author -licensetext -use -version -splitindex -noindex -nodeprecated -nodeprecatedlist -
nohelp -nonavbar -nosince -notree -public -protected -package -private -docfilessubdirs -
excludedocfilessubdir dirname -linksource
```

Output Options
: See Section 4.6 [Options Controlling the Output], page 25.

```
-d -locale name -charset charset -docencoding charset -validhtml -baseurl url
```

Decoration Options

```
-windowtitle text -doctitle text -title text -header text -footer text -
bottom text -helpfile file -stylesheetfile file -addstylesheet file -group group-
heading pkgpattern:pkgpattern:...
```

Taglet Options

See Section 4.7 [Options For Specifying user-defined Taglets], page 28.

```
-tagletpath -taglet classname -tag tagspec
```

Doclet Options

See Section 4.8 [Options For Specifying the Doclet to use], page 29.

```
-docletpath -doclet classname
```

Verbosity Options

See Section 4.11 [Options Controlling Gjdoc Behavior], page 32.

```
-quiet -verbose
```

Virtual Machine Options

See Section 4.12 [Options Controlling Gjdoc Behavior], page 32.

```
-classpath
-bootclasspath
-J
```

vmopt

4.3 Selecting which Source Files to Process

`-s` *pathlist*
`-sourcepath` *pathlist*

Look for source files in the specified directory or directories.

pathlist should be one or more directory paths separated by your platform's path separator (usually '`:`' or '`;`').

If this option is not given, `gjdoc` will look for source files in the current directory.

The directories specified should be root directories in terms of the Java package system. For example, if you want to generate documentation for classes in package '`foo.bar`', you must specify the directory containing the top-level ''`foo`'' sub-directory, not the directory ''`foo/bar/`'' in which the Java source files reside.

The short-hand alias '`-s`' is specific to `gjdoc` and not compatible to Sun `javadoc`.

`-all` *[EXPERIMENTAL]* Process all valid Java source files found in the directories listed in the source path and their sub-directories.

This is an option specific to `gjdoc` and not compatible to Sun `javadoc`.

`-subpackages` *pkg:pkg:...*

Process the classes in the given Java packages and all sub-packages, recursively. Note that multiple package names must be separated with colons instead of whitespace.

`-exclude` *`pkg:pkg:...`*
: Do not process classes in the given Java packages and all sub-packages, recursively. This option can be used in conjunction with '`-all`' or '`-subpackages`' in order to exclude individual packages or package sub-trees from the output.

`packages...`
: Process all classes in the given Java packages.

`sourcefiles...`
: Process the classes in the given Java source files.

4.4 Specifying the Format of Input Files

`-source` *`release`*
: Assume that the source files are targeted at the given release of the Java platform.

  release should be the version number of a Java platform release in the format MAJOR.MINOR, for example '`1.4`'.

  This option is currently ignored except that an error is raised if a release number other than '`1.2`', '`1.3`' or '`1.4`' is specified.

`-encoding` *`charset`*
: Assume that the source files are encoded using *charset*.

  Examples for *charset* are '`US-ASCII`', '`ISO-8859-1`' or '`UTF-8`'.

  The semantics of *charset* are identical to those of '`java.nio.charset.Charset.forName(String)`'.

`-breakiterator`
: Use the locale's java.text.BreakIterator instead of the internal first sentence detector.

  By default, `gjdoc` uses an internal algorithm to determine where a sentence ends. When this option is given, it will instead use the '`java.text.BreakIterator`' instance for the locale given with '`-locale`' (or the default locale).

  This option should be specified when applying `gjdoc` to source code commented in a non-latin language for which the default first sentence detector does not work. For all other cases, the default (do not use BreakIterator) produces better results at the time of this writing.

4.5 Interlinking with other Documentation Sets

`-link` *`url`*
: Create hyperlinks to another documentation set.

  By default, `gjdoc` will only create hyperlinks to classes in the source set. Use this option to additionally create hyperlinks to classes covered by the specified documentation set.

  url should be the root URL of the other documentation set. For example, to add hyperlinks to GNU Classpath, specify the following:

```
-link http://developer.classpath.org/doc/
```

The '`-link`' option can be specified multiple times.

Note that specifying the '`-link`' option will cause an HTTP access every time gjdoc is invoked. You can use '`-linkoffline`' instead to avoid this access.

`-linkoffline` *`url file`*

Create hyperlinks to another documentation set which is also present on the local file system.

This option works exactly like '`-link`', except that it accesses the local file system instead of the network for determining which classes are covered by the linked documentation set.

When using '`-linkoffline`' the remote documentation set is not accessed at all, which can significantly speed up generation time depending on your network connection. The generated hyperlinks to the documentation set however refer to the remote set, not to the local one, so that you can distribute the documentation without any further dependencies.

The '`-linkoffline`' option can be specified multiple times.

`-noqualifier` *`pkg:pkg:...`*

Do not qualify names of classes in the given packages with their package name.

By default, a class name is displayed unqualified only if the class is part of the source set or a linked documentation set, and qualified with the name of its containing package if it is not. You can use this option to force unqualified names for classes even if they are not part of the documentation set.

For example, usually a reference to the String class is represented fully-qualified as '`java.lang.String`' (unless you link to the appropriate documentation set using '`-link`') because it isn't part of the documentation set. You can specify '`-noqualifier java.lang`' to render the same references just as '`String`'.

Note that for all unqualified class names, a tooltip is provided when you place your mouse pointer over it in the HTML documentation.

`-noqualifier 'all'`

Omit package name qualifier from all class names.

Specify this option to omit package name qualifiers altogether,

4.6 Selecting which Information to Generate

`-public` Only include public members of public classes in the output. By default, protected class members are included as well.

`-protected`

Include public or protected members of public classes in the output. This is the default.

`-package`

Include public, protected and package-private members of public and package-private classes.

`-private`
: Include all classes and class members regardless of their access level.

`-splitindex`
: Generate one index page per letter instead of a single, monolithic index page.

  By default, the index created by the Standard Doclet contains all entries on a single page. This is fine for small documentation sets, but for large sets you should specify this option.

`-nosince` Ignore '`@since`' tags in javadoc comments.

  By default, the generated output contains sections listing the version of your API since which the package, class or class member in question exists when this tag is encountered. Specify this option to omit this information.

`-notree` Do not generate any tree pages.

  By default, the generated output includes one inheritance tree per package, and - if the documentation set consists of multiple packages - a page with the full inheritance tree. Specify this option to omit generation of these pages.

`-noindex` Do not output the alphabetical index.

  By default, gjdoc generates an alphabetical index of all program elements in the documentation set (packages, classes, inner classes, constructors, methods, and fields). Specify this option to omit this information.

`-nohelp` Do not generate the help page.

  This option is currently ignored as the Standard Doclet doesn't provide a help page.

`-nodeprecated`
: Do not output inline information about deprecated packages, classes or class members.

  By default, the Standard Doclet adds a highlighted paragraph with deprecation information to the description of each deprecated program element. Specify this option to omit this information.

`-nodeprecatedlist`
: Do not output the summary page for deprecated API elements.

  By default, the Standard Doclet generates a page listing all deprecated API elements along with a deprecation description which usually includes the reason for deprecation and possible alternatives. Specify this option to omit this information.

`-nonavbar`
: Do not output the navigation bar, header, and footer.

  By default, each output page is equipped with a top navigation bar (which may include a user-specified header) and a bottom navigation bar (which may include a user-specified footer). Specify this option to omit this decoration.

`-nocomment`
: Omit all documentation text from the generated files and output only declarations and program element relationships.

This option is here for compatibility with `javadoc`. If you plan on extracting information about your project via `gjdoc`, you should consider using a different Doclet for your purposes instead, for example XmlDoclet. You could also use the Doclet API directly by implementing a new Doclet.

`-linksource`
: Generate a page with syntax-highlighted source code for each class. By default, this page is not generated.

  The source code can be accessed by clicking on the button labelled `"Source"` in the navigation bar, or by clicking on the name of a constructor, field, method, or inner class in the detail section of a class documentation page.

`-use`
: Generate a page with cross-reference information. By default, this page is not generated.

  The cross-reference information can be accessed by clicking on the button labelled 'Use' in the navigation bar.

  The 'Use' page lists all classes/interfaces in the documentation set that extend/implement the class (type) in question; fields of the type; methods or constructors accepting a parameter of the type; methods returning the type; and methods or constructors throwing the type.

`-author` Include author information in the output.
: When specified, author information as specified using the '`@author`' tag in javadoc comments is incorporated into the output. By default, '`@author`' tags are ignored.

`-version` Include version information in the output.
: When specified, version information as specified using the '`@version`' tag in javadoc comments is incorporated into the output. By default, '`@version`' tags are ignored.

`-licensetext`
: Assume that the first comment in each source file contains the license text, and add license information to the footer of each generated class page.

  This is an option specific to `gjdoc` and not compatible to Sun `javadoc`.

  This option is intended for use with free and open source projects where source code is typically prefixed with a boilerplate license comment, when there are legal reasons for including the license in the documentation.

`-docfilessubdirs`
: Recursively copy all files in the '`doc-files`' sub-directory of each package directory.

  Usually, only the files in the '`doc-files`' sub-directory are copied without descending recursively.

  See Section 6.5 [Adding Custom Resources], page 38.

`-excludedocfilessubdir` *`name`*`:`*`name`*`:...`
: Do not copy some directories directly under the '`doc-files`' sub-directories when descending recursively.

The argument to this option should be a colon-separated list of directory names.

This option only makes sense if '`-docfilessubdirs`' is also specified. In this case, any sub-directory located directly beneath a '`doc-files`' directory is omitted if listed.

4.7 Custom Documentation Tags

`-tagletpath` *pathlist*

Search *pathlist* when loading subsequent Taglet classes specified using '`-taglet`'.

pathlist should be one or more paths to a directory or jar file, separated by your platform's path separator (usually '`:`' or '`;`').

`-taglet` *classname*

Register a Taglet.

classname should be the fully-qualified name of a Java class implementing '`com.sun.tools.doclets.Taglet`'.

The Taglet classes will be loaded from the classpath specified using '`-tagletpath`', from the classpath specified using '`-classpath`' and from the default classpath.

See the documentation of '`com.sun.tools.doclets.Taglet`' for further information.

Note that for simple tags, there is also '`-tag`'.

`-tag` *tagspec*

Register a generic Taglet.

The format of *tagspec* must be '`<tagname>:<flags>:"<taghead>"`'.

tagname is the tag name to match, without the leading `@` sign.

flags is one or more of the following characters, where each character specifies a source code context in which the tag is to be recognized.

`a`	all contexts
`c`	constructors
`f`	fields
`m`	methods
`o`	overview
`p`	packages
`t`	types (classes, interfaces, exceptions, errors)
`X`	special character which temporarily disables the Taglet altogether.

taghead is the string to display in the header of the section devoted to the tag in question.

For example, to define a tag matching '`@cvsid`' which is to be accepted in overview, package and type pages and which is labelled with the header '`CVS ID`', you would specify:

```
-tag cvsid:tpo:"CVS ID"
```

Let's say that a class javadoc comment contains

```
@cvsid $Id: cp-tools.texinfo,v 1.7 2008/08/13 13:32:05 jsumali Exp $
```

Then the HTML output will contain something like

```
CVS ID:
  $Id: cp-tools.texinfo,v 1.7 2008/08/13 13:32:05 jsumali Exp $
```

4.8 Running Other Doclets

`-docletpath` *pathlist*

Search *pathlist* when loading classes for the Doclet specified using '`-doclet`'.

pathlist should be one or more paths to a directory or jar file, separated by your platform's path separator (usually '`:`' or '`;`').

`-doclet` *className*

Run the specified doclet instead of the standard HtmlDoclet.

className should be the fully-qualified name of a class which has a public default constructor and contain a method with the following signature:

```
import com.sun.javadoc.RootDoc;
public static boolean start(RootDoc rootDoc)
```

The Doclet classes will be loaded from the classpath specified using '`-docletpath`', from the classpath specified using '`-classpath`' and from the default classpath.

The '`start`' method should process the information exposed by the Doclet API via '`rootDoc`' and return '`true`' on success, '`false`' on failure.

If you are using a third-party doclet, refer to its documentation for further instructions. Note that support for third-party doclets is experimental. Please report any problems you encounter, or provide feedback when successfully running third-party applets.

This option can be specified multiple times, in which case all doclets are executed with the same information tree exposed via the Doclet API for each Doclet run.

4.9 Adding Information to the Output

`-windowtitle` *text*

Use *text* as the browser window title prefix.

When specified, the browser window title for each page will be prefixed with *text* instead of the default string '`Generated API Documentation`'.

text should be plain text (it should not contain HTML tags).

`-doctitle` *text*

Set the header text of the overview page to *text*.

text should be a short plain text string.

When generating documentation for a single package, specifying this option forces generation of the overview page.

`-header` *htmltext*
: Add *htmltext* to the right upper corner of every generated page. *htmltext* is usually set to the name of the project being documented.

`-footer` *htmltext*
: Add *htmltext* to the right bottom corner of every generated page. *htmltext* is often set to the same value as for '`-header`'.

`-bottom` *htmltext*
: Add *htmltext* to the very bottom of every generated page, spanning the whole width of the page. When specified, *htmltext* usually consists of a copyright notice and/or links to other project pages.

`-addstylesheet` *file*
: Augment the default CSS style sheets with the user-specified stylesheet *file*.

  The given stylesheet is simply loaded by each HTML page in addition to the default ones, as the last stylesheet.

  Note that the CSS cascading rules apply. That is, your style properties will only be assigned if they have a higher cascading order than `gjdoc`'s default style. One simple way to make sure that this is the case is to declare your overrides '`!important`'.

  See http://www.w3.org/TR/REC-CSS2/cascade.html#cascading-order.

`-group` *heading pkgwildcard*`:`*pkgwildcard*`:...`
: Arrange the given packages in a separate group on the overview page.

  The first argument should be a short plain text which is used as the title of the package group. The second argument should be a colon-separated list of package wildcards. The group will consist of all packages in the documentation set whose name matches any of the given wildcards.

  There is only one wildcard character, '`*`', which matches both letters in package name components and the '`.`' separating package name components. For example, '`j*regex`' would match package '`java.util.regex`'. A more useful example would be '`javax.swing*`' to match '`javax.swing`' and all of its subpackages.

  This option can be given multiple times.

  FIXME: Information about group nesting here.

```
gjdoc -group "Core Classes" 'java*' \
      -group "Swing" 'javax.swing*' \
      -group "XML APIs" 'javax.xml*' \
      -group "Other Extensions" javax* \
      ...
```

`-overview` *file*
: Add the XHTML body fragment from *file* to the overview page.

  file should contain an XHTML fragment with the HTML '`body`' tag as the root node. See Section 6.3 [XHTML Fragments], page 37.

  This option can be used to supply a description of the documentation set as a whole.

When specified, the first sentence of the fragment will be put above the tables listing the documented packages, along with a link to the full copy of the fragment which is put below the tables. See Section 6.4 [First Sentence Detector], page 37.

When generating documentation for a single package, specifying this option forces generation of the overview page.

`-stylesheetfile` *file*
: Use the CSS stylesheet in *file* instead of the default CSS stylesheets.

  If you only want to override parts of the default stylesheets, use '`-addstylesheet`' instead.

`-title` *text*
: *Deprecated.* Use '`-doctitle`' *text* instead.

`-helpfile` *file*
: This option is currently ignored.

  When implemented, it will use the XHTML fragment in *file* for the help page contents instead of the default help text.

4.10 Controlling the Output.

`-d` *directory*
: Place all output files into *directory* (and sub-directories). *directory* will be created if it does not exist, including all non-existing parent directories and all required sub-directories.

  If not specified, output will be placed into the current directory.

`-locale` *name*
: Use locale *name* instead of the default locale for all purposes.

  name should be a locale specifier in the form '`ll_CC[_VAR]`' where '`ll`' is a lowercase two-letter ISO-639 language code, '`CC`' is an optional uppercase two-letter ISO-3166 country code, and '`VAR`' is an optional variant code. For example, '`en`' specifies English, '`en_US`' specifies US English, and '`en_US_WIN`' specifies a deviant variant of the US English locale.

  Note that the semantics of this option correspond exactly to those of the constructors of class '`java.util.Locale`'.

  This option currently only determines which Collator is being used for sorting output elements. This means that the locale will only have an effect when you are using non-ASCII characters in identifiers.

`-charset` *charset*
: *Deprecated.* Override the specified encoding in output XHTML files with the one given by '`charset`'.

  If this option is not given, the encoding specification in output XHTML is chosen to match the encoding used when writing the file (the encoding given with '`-docencoding`', or your platform's default encoding).

The semantics for *charset* are specified here: http://www.w3.org/TR/2000/REC-xml-20001006#NT
For all practical purposes, they are identical to those of the other options accepting charset parameters.

This option is here for compatibility with `javadoc` and should be avoided.

`-docencoding` *charset*
: Use the given charset encoding when writing output files instead of your platform's default encoding.

  Examples for *charset* are '`US-ASCII`', '`ISO-8859-1`' or '`UTF-8`'.

  The semantics of this option correspond exactly to those of the constructors of class '`java.util.Locale`'.

`-validhtml`
: Force generation of valid XHTML code. This breaks compatibility to the traditional Javadoc tool to some extent.

  If this option is specified, anchor names will be mangled so that they are valid according to the XHTML 1.1 specification. However, a documentation set generated with this option cannot be linked to properly using the traditional Javadoc tool. It can be linked to just fine using Gjdoc, though.

  Without this option, anchor names for executable class members use the traditional format, for example: "foo(String,int[])". This is compatible to the traditional Javadoc tool, but according to both the HTML 4.0 and XHTML 1.0 and 1.1 specifications, this format includes illegal characters. Parentheses, square brackets, and the comma are not allowed in anchor names.

`-baseurl` *url*
: Hardwire a page URL relative to *url* into each generated page.

  If you are generating documentation which will exclusively be available at a certain URL, you should use this option to specify this URL.

  This can help avoid certain redirect attacks used by spammers, and it can be helpful for certain web clients.

4.11 Verbosity Options

`-quiet` Suppress all output except for warnings and error messages.

`-verbose` Be very verbose about what `gjdoc` is doing.

This option is currently ignored.

4.12 Virtual Machine Options

Sun's `javadoc` tool seems to be based on `javac` and as such it seems to operate on the VM level. `gjdoc`, in contrast, is a pure Java application.

Therefore, `gjdoc` can only fake, or simulate, the following VM-level options.

`-classpath` *pathlist*
: Set the Virtual Machine '`classpath`' to *pathlist*.

  In most cases you should use '`-docletpath`' or '`-tagletpath`' instead of this option.

pathlist should be one or more paths to a directory or jar file, separated by your platform's path separator (usually '`:`' or '`;`').

If this option is not intercepted at the wrapper level, `gjdoc` currently fakes it by calling '`System.setProperty("java.class.path", pathlist);`' and outputs a warning.

`-bootclasspath` *pathlist*

Set the Virtual Machine '`bootclasspath`' to *pathlist*.

If this option is not intercepted at the wrapper level, `gjdoc` outputs a warning.

`-J`*vmopt*

Pass an arbitrary parameter to the Virtual Machine `gjdoc` runs on.

If this option is not intercepted at the wrapper level, `gjdoc` tries to emulate the option and outputs a warning.

Currently, only the VM option '`-D`' for setting system properties is emulated.

4.13 Invoking a Custom Doclet

For invoking one of the other doclets shipping with `gjdoc` or a third-party doclet, the canonical usage is:

```
gjdoc -s src/java/ -all \
  -docletpath /path/to/doclet.jar -doclet foo.BarDoclet \
  (more Gjdoc core options and Doclet-specific options here)
```

'`/path/to/doclet.jar`' is a placeholder for a class path specifying where the Doclet classes and dependencies can be found and '`foo.BarDoclet`' is the fully-qualified name of the Doclet's main class.

4.14 Gjdoc Option Summary

5 Generating Other Output Types

5.1 Using the Built-in Doclets

5.1.1 TexiDoclet: Generating Info, PDF, and other formats

Missing.

5.1.2 XmlDoclet: Generating XML Documentation

Missing.

5.1.3 IspellDoclet: Spell-checking Source Code

Missing.

5.1.4 DebugDoclet: Inspecting the Doclet API

Missing.

5.2 Using Third-Party Doclets

5.2.1 DocBook Doclet

Missing.

5.2.2 PDFDoclet

Missing.

5.2.3 JUnitDoclet

Missing.

6 Advanced Concepts

6.1 Adding Custom Tags to the Documentation

Missing.

6.2 Writing Doclets

If the various Doclets already available don't suit your needs, you can write a custom Doclet yourself.

6.2.1 Implementing the Doclet Invocation Interface

A Doclet is a class that contains a method with the following signature:

```
public static boolean start(RootDoc rootDoc);
```

rootDoc is the root of an object hierarchy containing the information `gjdoc` extracted from the source files. See the Doclet API for more details.

'`start`' should process all the information and return '`true`' on success, '`false`' on failure.

For printing status information, the Doclet should use methods '`printNotice`', '`printWarning`' and '`printError`' instead of '`System.err`'. The Doclet can opt to use '`System.out`' for redirectable output.

6.2.2 Deriving Your Doclet from AbstractDoclet

You may want your Doclet to provide functionality similar to HtmlDoclet. For example, you may want it to support Taglets, generate Index, Tree, and Uses pages, or show other cross-reference information like '`Overrides`' and '`All Implementing Classes`'.

This information is not directly provided by the Doclet API, so your Doclet would normally have to assemble it itself. For example, it would have to add the names of all program elements to a list and sort this list in order to create the Index page.

If you want to provide this information or part of it, you should consider deriving your class from '`gnu.classpath.tools.doclets.AbstractDoclet`'. This class provides the following benefits:

- Handles options '`-tag`', '`-taglet`', '`-tagletpath`' (Taglets)
- Provides standard taglets for `@`version, `@`author, `@`since, `@`serial, `@`deprecated, `@`see, `@`param, `@`return and handles all related options ('`-version`', '`-author`', '`-nosince`', '`-nodeprecated`')
- Handles option '`-d`' (destination directory)
- Handles option '`-noqualifier`' (classes to omit qualifier for)
- Handles options '`-docfilessubdirs`' and '`-excludedocfilessubdir`' (resource copying)
- Can generate a full index or an index split by first letter
- Can generate a full tree and package trees
- Can generate cross-reference information

- Can aggregate interface information (all superinterfaces, all subinterfaces, all implementing classes)
- Provides convenient access to constructors, fields, methods, and inner classes sorted by name/signature instead of the default sort order.
- Provides various other convenience methods

If you derive from '`AbstractDoclet`', there are a number of things you need to take care of:

-

you should not implement the '`start(RootDoc)`' method as it is already defined by '`AbstractDoclet`' so that it can care about parsing the options.

Instead, you implement method '`run()`', '`getOptions()`' and the other abstract methods to define your Doclet's behavior.

Note that all information provided by '`AbstractDoclet`' is evaluated lazily. That is, if your Doclet doesn't need to create an Index page, then '`AbstractDoclet`' will not spend resources on creating the corresponding information.

See the API documentation for '`gnu.classpath.tools.doclets.AbstractDoclet`' for more details.

You should be aware that if you base your Doclet on '`AbstractDoclet`' then you have to bundle this and all related classes with your Doclet, with all implications such as possible licensing issues. Otherwise, your Doclet will only be runnable on '`gjdoc`' and not on other documentation systems. Also note that '`AbstractDoclet`' has not been extensively tested in environments other than '`gjdoc`'.

6.2.3 Preparing for the GNU Doclet Service Provider Interface

In addition to the standard Doclet invocation interface described above, `gjdoc` also offers a Service Provider Interface conforming to the Java standard. Adding support for this interface to your Doclet simplifies usage for `gjdoc` users because it makes your Doclet "discoverable".

In order to provide the alternate interface, you have to add a class implementing '`gnu.classpath.tools.gjdoc.spi.DocletSpi`' to your Doclet classes, and bundle all Doclet classes in a Jar file along with a file named '`META_INF/services/gnu.classpath.tools.gjdoc.spi.DocletSpi`' which contains the name of your class implementing DocletSpi on a single line.

Note that if your Doclet depends on third-party classes bundled in separate Jar files, you can link in these classes using the '`Class-path:`' Manifest attribute of your Doclet Jar.

Your Doclet can then be invoked in one of the following ways:

```
gjdoc -docletjar /path/to/doclet.jar
gjdoc -docletpath /path/to/doclet.jar -docletname docletname
gjdoc -docletname docletname
```

Here, *docletname* is the name of your doclet as returned by '`DocletSpi.getDocletName()`'.

The last example will only work if your Doclet Jar is in `gjdoc`'s '`doclets`' directory or if it is on the classpath.

6.3 Well-formed Documentation Fragments

For many Doclets it is advantagous if the HTML code in the comments and HTML code passed via the command line is well-formed. For example, HtmlDoclet outputs XHTML code, and XmlDoclet XML code, both of which results in invalid files if the user-specified HTML isn't wellformed.

Unfortunately, comments were never required to contain well-formed HTML code, which means that every Doclet must deal with non-wellformed code as well.

The `gjdoc` built-in Doclets deal with this problem by "fixing" the HTML code - making sure that all tags are closed, attribute values are provided and quoted, tags are properly nested, etc.

This approach works OK in most instances, but since it uses some crude heuristics it can sometimes produce undesirable result.

Therefore, in order to make sure that your comments are always properly formatted, make sure they are well-formed as described in XHTML 1.0: Documents must be well-formed.

In addition, you should use meaningful tags instead of text formatting tags to make your output look better in other output formats derived from your HTML code. For example, you should use the tag instead of <b> if you want to emphasize text.

6.4 How Gjdoc Determines where the First Sentence Ends

For a package, class or member summary, `gjdoc` only shows the first sentence of the documentation comment in question. Because `gjdoc` is not human, it is not always obvious to `gjdoc` where the first sentence ends.

You might be tempted to say that the first sentence ends at the first occurrence of a punctuation character like '`.`' or '`!`'. However, consider examples like this:

```
This work, by Thomas J. Shahan et al., is about the middle ages.
```

As you can see, it is not trivial to determine the end of the sentence.

`gjdoc` gives you the choice between two approaches. By default it uses built-in heuristics which should be compatible to Sun's `javadoc` tool. This approach works quiet well in most cases, at least for english comments.

Alternatively, you can specify option '`-breakiterator`' in which case `gjdoc` will use '`java.text.BreakIterator.getSentenceInstance(`*`locale`*`).next()`' to find the end of sentence, where *locale* is the locale specified by option '`-locale`' or the default locale if none specified.

NOT YET IMPLEMENTED:

`gjdoc` also allows you to explicitly delineate the first sentence by putting it in a '`<span>`' tag with the CSS class '`first-sentence`'. For example:

```
/**
 *  <span class="first-sentence">This. is. the. first.
 *  sentence.</span> This is the second sentence.
 */
```

Note that this will only work with `gjdoc`, but shouldn't hurt when using another documentation system since the '`<span>`' tag usually doesn't show up in the output.

6.5 Adding Images and Other Resources

Sometimes you want to decorate your documentation with secondary resources such as images, SVG graphics, applets, and so on. To do so, simply put the required files in a subdirectory 'doc-files' in the package directory corresponding to the documentation entry you want to decorate, and refer to it with the URL '`doc-files/`*`filename`*'.

For example, if you want to add an image to the description of class '`baz.FooBar`', create a directory '`doc-files`' in the directory '`baz`' containing '`FooBar.java`' and put your file, say '`diagram.png`', into that directory. Then, add the HTML code like this to a comment in '`FooBar.java`':

```
<img src="doc-files/diagram.png" width="200" height="150"
  alt="Foo Diagram"/>
```

This works because the '`doc-files`' subdirectories will be copied to the target documentation directory every time you generate the documentation.

Note however that by default, the '`doc-files`' directory will not be copied deeply. In other words, if you create subdirectories under '`doc-files`' they will not be copied and any resources located in these subdirectories will not be accessible in your generated documentation. You can specify option '`-docfilessubdirs`' to remove this limitation.

Sometimes you want to use option '`-docfilessubdirs`', but there are certain directories which you don't want to be copied, for example because they contain source files for the resources in '`doc-files`'. For cases like this, use '`-excludedocfilessubdir`' to specify directories to be omitted.

7 I18N Issues

Some tools –see Chapter 2 [Security Tools], page 3– allow using other than the English language when prompting the User for input, and outputting messages. This chapter describes the elements used to offer this support and how they can be adapted for use with specific languages.

7.1 Language-specific resources

The Tools use Java `ResourceBundle`s to store messages, and message templates they use at runtime to generate the message text itself, depending on the locale in use at the time.

The *Resource Bundles* these tools use are essentially Java *Properties* files consisting of a set of *Name/Value* pairs. The *Name* is the *Property Name* and the *Value* is a substitution string that is used when the code references the associated *Name*. For example the following is a line in a *Resource Bundle* used by the `keytool` Tool:

```
Command.23=A correct key password MUST be provided
```

When the tool needs to signal a mandatory but missing key password, it would reference the property named `Command.23` and the message "*A correct key password MUST be provided*" will be used instead. This indirect referencing of "resources" permits replacing, as late as possible, the English strings with strings in other languages, provided of course *Resource Bundles* in those languages are provided.

For the GNU Classpath Tools described in this Guide, the *Resource Bundles* are files named '`messages[_ll[_CC[_VV]]].properties`' where:

ll Is the 2-letter code for the Language,

CC Is the 2-letter code for the Region, and

VV Is the 2-letter code for the Variant of the language.

The complete list of language codes can be found at Code for the representation of names of languages. A similar list for the region codes can be found at ISO 3166 Codes (Countries).

The location of the *Resource Bundles* for the GNU Classpath Tools is specific to each tool. The next table shows where these files are found in a standard GNU Classpath distribution:

`jarsigner`
: '`gnu/classpath/tools/jarsigner`'

`keytool` '`gnu/classpath/tools/keytool`'

The collection of *Resource Bundles* in a location act as an inverted tree with a parent-child relationship. For example suppose in the '`gnu/classpath/tools/keytool`' there are 3 message bundles named:

1. `messages.properties`
2. `messages_fr.properties`
3. `messages_fr_FR.properties`

In the above example, bundle #1 will act as the parent of bundle #2, which in turn will act as the parent for bundle #3. This ordering is used by the Java runtime to choose which file to load based on the set Locale. For example if the Locale is `fr_CH`, `messages_fr.properties` will be used because (a) `messages_fr_CH.properties` does not exist, but (b) `messages_fr.properties` is the parent for the required bundle, and it exists. As another example, suppose the Locale was set to `en_AU`; then the tool will end up using `messages.properties` because (a) `messages_en_AU.properties` does not exist, (b) `messages_en.properties` which is the parent for the required bundle does not exist, but (c) `messages.properties` exists and is the root of the hierarchy.

You can see from the examples above that '`messages.properties`' is the safety net that the Java runtime falls back to when failing to find a specific bundle and its parent(s). This file is always provided with the Tool. In time, more localized versions will be included to cater for other languages.

In the meantime, if you are willing to contribute localized versions of these resources, grab the '`messages.properties`' for a specific tool; translate it; save it with the appropriate language and region suffix and mail it to `classpath@gnu.org`.

7.2 Message formats

If you open any of the '`messages.properties`' described in the previous section, you may see properties that look like so:

```
Command.67=Issuer: {0}
Command.68=Serial number: {0,number}
Command.69=Valid from: {0,date,full} - {0,time,full}
Command.70=\ \ \ \ \ until: {0,date,full} - {0,time,full}
```

These are *Message Formats* used by the tools to customize a text string that will then be used either as a prompt for User input or as output.

If you are translating a '`messages.properties`' be careful not to alter text between curly braces.