Logging FrequentlyAskedQuestions

Commons Logging FAQ

Space for questions and answers. Note that the user mailing list is usually the best place to get answers to question. Remember to search the archives!

If someone on the mailing lists has answered your question, why not save everyone else a lot of trouble by recording the answer here 🙂

When should I use commons-logging?

Commons-logging is very important when writing **libraries** that might be used in other projects. If the library directly calls a concrete logging library API (eg log4j or java.util.logging) but the application it is embedded in is using some other logging library then there is a problem. However if the library is using commons-logging then commons-logging simply forwards all calls to whatever logging library the surrounding application is using, ensuring output is all nicely unified. The specific underlying library commons-logging uses can be explicitly configured (recommended), or commons-logging can be permitted to auto-detect it.

Commons-logging can also be useful when you are writing an **application** but want to allow the person installing your app to choose what logging lib to use. Apache Tomcat does this; the person installing it can choose to use log4j, java.util.logging, etc. There is some debate over how useful this actually is.

Commons-logging can also be useful when you are writing an **application** but want to reserve the right to swap logging libraries at a later date. This is not so important, though, as it really isn't that hard to use some refactoring tool or a few scripts to change source code from using library X to library Y.

What Java versions are supported?

Commons-logging versions 1.0.2 through 1.0.4 require java 1.2. Work is currently in progress on the successor to 1.0.4, and thought is being given to restoring support for java version 1.1 - it is not currently known whether this is possible.

Commons-logging version 1.0.1 is believed to work with java 1.1, but only in Applets. To be more specific, commons-logging 1.0.1 is believed to fail on java 1.1 when the commons-logging classes are loaded by the system classloader; this *is* the case for normal applications, but not the case for Applets.

If you encounter difficulties using commons-logging on specific java versions, please post to the commons-user list so the problem can be addressed and /or this FAQ entry updated.

What alternatives are there to commons-logging?

The SLF4J project has similar goals to commons-logging. It uses quite a different approach, however, which has both benefits and drawbacks. Future versions of commons-logging may adopt some of the concepts of SLF4J. See http://slf4j.org for more details.

Log Bridge (https://log-bridge.dev.java.net/) is very similar to commons-logging, the main differences being the inclusion of entry() and exit() methods (instead of a 'trace' level), programmatic configuration, "less magic" (no auto-selection of factories) and support for Simple Log (https://simple-log.dev.java.net/).

The Avalon LogKit provides adapters to other logging libraries. However this library is not very widely used and is not (as far as I know) maintained any longer.

The JULI module in Apache Tomcat provides adapters from the java.util.logging API. This is of course not an option for code that may be run in JVMs prior to 1.4. The java.util.logging system can also be awkward to set up.

Can calls to java.util.logging be redirected via commons-logging?

Yes. The java.util.logging classes present in java since 1.4 are both an API and a (primitive) logging implementation. It is possible to install an "implementation" that redirects messages back to commons-logging, which will then in turn direct the calls to the appropriate concrete logging library that commons-logging is sending other messages to.

Alternatively, have your java.util.logging "implementation" send messages directly to the same implementation that commons-logging is bound to. This will be faster - although if you change your commons-logging configuration to use a different logging library then the java.util.logging implementation would need to be changed too.

See here for details:
http://wiki.apache.org/myfaces/Trinidad_and_Common_Logging

Is JCL Thread Safe?

JCL doesn't (and cannot) impose any requirement on thread safety on the underlying implementation and thus its SPI contract doesn't guarantee thread safety. However, JCL can be safely used in a multi-threaded environment as long as the underlying implementation is thread-safe.

It would be very unusual for a logging system to be thread unsafe. Certainly, JCL is thread safe when used with the distributed Log implementations.

How Can I Switch Logging Levels On And Off?

See How Do I Change The Logging System Configuration?

How Do I Change The Logging System Configuration?

The configuration supported by JCL is limited to choosing the underlying logging system. JCL does not (and will never) support changing the configuration of the wrapped logging system. Please use the mechanisms provided by the underlying logging system.

The only exception is when using the SimpleLog logging facility built into commons-logging. In this case, see the documentation for that class for details on configuring logging levels.

Should I declare Log references static or not?

Short answer: static references to Log instances are ok for application code, but a bad idea for library code.

See this page for details:
http://wiki.apache.org/jakarta-commons/Logging/StaticLog

Log4JLogger does not implement Log

```
I have an exception with message 'Log4JLogger does not implement Log'!
What's the cause and how can I fix this problem?
```

It's due to classloaders. The error message should really read something like

```
  org.apache.commons.logging.impl.Log4JLogger loaded via classloader XXXXX
  is bound to interface org.apache.logging.impl.Log as loaded via classloader XXXXX
  but is required to bind to org.apache.logging.impl.Log as loaded by classloader YYYYY.
```

If you get this problem, please post to the commons-user list. We would love to hear from you so that we can ask you to run some simple diagnostic code. We really need more detailed information on this issue, including things like

- what is the classloader hierarchy
- whether parent-first or parent-last (aka child-first) classloading is selected
- exactly which classloader has ended up loading which logging-related classes

There are some things you can try to resolve this issue. Ensure that:

- All the logging classes (both Log and the logging implementations) are deployed by the same classloader. In other words, place commons-logging.jar and the logging library jar in the same directory.
- There is only one copy of the classes to be found within the classloader hierarchy. In application container environments this means ensuring that if the classes are found in a parent classloader, they are not also present in the leaf classloader associated with the application. So, if the jar is deployed within the root classloader of the container then it should be removed from the application's library.

Known causes of "does not implement Log"

Running JspC from Ant

The Tomcat JspC tool compiles JSP pages into bytecode. It had a bug in versions prior to 5.5.5 or 5.0.30. where it set the context classloader to a non-standard value, then failed to reset it to the original value on exit. When run "in-process" as an ant task this would cause commons-logging to try to load logging libraries from an invalid classloader. This bug has been fixed in the 5.5.5 and 5.0.30 releases of Tomcat.

How Can I Use Commons-Logging In WebSphere 5.1?

```
I'm using WebSphere 5.1. Commons-Logging doesn't seem to recognize my commons-logging.properties file. Help!
```

WebSphere uses commons-logging and so it's in the root classloader. This may cause difficulties when trying to find a configuration resource. The solution is to ensure that the right classloader policy is set:

```
Set EAR classloader mode as PARENT_LAST and WAR classloader policy as Application
```

(With thanks to David Karlsen for this information)

For more information see http://mail-archives.apache.org/eyebrowse/ReadMsg?listName=commons-user@jakarta.apache.org&msgNo=5737

How Can I Use Log4j with Commons-Logging 1.1 on WebSphere 6.0?

```
I'm using WebSphere 6.0, Commons Logging 1.1 and Log4j. I have the log4j and commons-logging jars on my
classpath, but my log4j logging is not working. Help!
```

WebSphere uses commons-logging and so it's in the root classloader. In addition, WebSphere 6.0 ships a commons-logging.properties with the following properties set:

```
org.apache.commons.logging.LogFactory=org.apache.commons.logging.impl.LogFactoryImpl
org.apache.commons.logging.Log=org.apache.commons.logging.impl.Jdk14Logger
```

Therefore by default the application will use JDK logging and not log4j. The solution is to ensure that the right classloader mode is set and the application has an appropriate commons-logging.properties:

```
Set application classloader mode as PARENT_LAST. Also, add a commons-logging.properties to the application
classpath with the following entries:

priority=1
org.apache.commons.logging.LogFactory=org.apache.commons.logging.impl.LogFactoryImpl
```

The priority flag was introduced in Commons Logging 1.1 to allow an ordering based on priority. To ensure that an application's commons-logging. properties will take precedence over WebSphere's file, a priority of greater that 0.0 must be set.

If you still have issues with Commons-Logging on WebSphere 5.x/6.0?

If you have still have issues with Commons Logging after implementing the changes discussed above, refer to IBM technote 'Commons-logging.properties not found on classpath' at http://www-1.ibm.com/support/docview.wss?uid=swg21211020. It works for sure.

I don't want to change my classloader behaviour to use Commons-Logging on WebSphere 5.x/6.0, is there any other way ?

There is also a workaround explained at 'http://www.ibm.com/developerworks/forums/thread.jspa?threadID=192621&tstart=0&messageID=14254344#14254344'

How Can I Use Commons-Logging In JBoss 4.0.2?

```
I'm using JBoss 4.0.2. Commons-Logging doesn't seem to recognize the commons-logging.properties file in my WAR.
Help!
```

Some JBoss services use commons-logging, so it's on the server classpath. Copies of commons-logging loaded from a webapp's WEB-INF/lib folder were causing type conflicts with the version on the server classpath, so beginning with version 4.0.2 JBoss disables loading commons-logging from a WAR. This has the side effect of preventing commons-logging from accessing logging libraries that are placed in the WEB-INF/lib folder.

Fortunately, there is an easy fix:

```
Place your logging library jar (e.g. avalon-framework.jar) in JBoss' server/.../lib folder (where "..." is
"all" or "default" or a custom configuration you use).
```

This places your logging library on the server classpath, where the copy of commons-logging used by JBoss can see it. The downside to this is your logging library is no longer hot-deployable, so upgrading it will require a server restart.

How can I close loggers when using Commons-Logging?

You can't.

Commons-logging is a wrapper intended to isolate code from the underlying logging library when log calls are being made. In particular, it is intended to allow library code to contain logging calls without caring about what concrete log implementation is being used by the wider application the library is deployed within.

It is not intended to isolate applications from the logging-library-specific details of initialization and termination.

The fact that some logging implementations auto-configure themselves when first used doesn't mean that commons-logging guarantees to do this in all cases, nor that the default initialization behaviour of the concrete logging libraries is appropriate under all circumstances.

And the fact that some logging libraries (or subsets thereof) terminate nicely without requiring library-specific shutdown code doesn't mean that all libraries will work correctly in this manner, nor that commons-logging will isolate code from this task.

In short: when necessary, obtain a reference to the actual underlying log implementation and use library-specific calls to initialise and terminate as required. Yes this means that at least one part of your code will need to know what actual logging library is being used, and reference classes from that actual logging library to do the appropriate initialisation/shutdown.

If you need a reference to the underlying object for a particular commons-logging Log object in order to perform initialisation or termination, then in most cases the org.apache.commons.logging.Log class can be downcast to a concrete type that then has a "getLogger" method that returns the real underlying object. However in most cases this is not necessary, as initialisation/shutdown methods need to be invoked on "the logging library as a whole" rather than the objects representing individual logging categories.

A memory leak occurs when undeploying/redeploying a webapp that uses Commons Logging. How do I fix this?

When commons-logging is deployed in a "shared" classloader that is visible to all webapps, commons-logging still ensures that each webapp gets its own logging configuration by setting up a separate LogFactory object for each context classloader, and ensuring that the correct LogFactory object is used based on the context classloader set whenever LogFactory.getLog is called.

However the negative side of this is that LogFactory needs to keep a static map of LogFactory objects keyed by context classloader; when the webapp is "undeployed" this means there is still a reference to the undeployed classloader preventing the memory used by all its classes from being reclaimed.

The commons-logging-optional.jar optional code provided with commons-logging 1.0.5 attempted to address this issue using weak references, but this approach cannot solve the issue in many cases.

The correct solution in the case of a webapp is for the webapp to include a class which implements ServletContextListener:

```
class ContextManager implements javax.servlet.ServletContextListener {
  public void contextDestroyed(ServletContextEvent sce) {
    ClassLoader contextClassLoader = Thread.currentThread().getContextClassLoader();
    LogFactory.release(contextClassLoader);

    // Optionally, clean up the underlying concrete logging library too. This might
    // require direct calls to methods on library-specific methods.

    // And optionally call java.beans.Introspector.flushCaches if your app does any
    // bean introspection and your container doesn't flush the caches for you on
    // servlet undeploy. Note that this isn't a commons-logging problem; it's something
    // quite unrelated which can also cause memory leaks on undeploy.
  }
}
```

The deployment descriptor for the webapp then needs to be updated to specify that the above class should receive context change callbacks.

You can find more information on this topic here.

I implemented Log, why do I get an exception saying that class is not useable?

When providing an implementation of the org.apache.commons.logging.Log interface, the implementation must define a constructor that takes a java.lang.String as an argument. Otherwise you get an "org.apache.commons.logging.LogConfigurationException: User-specified log class '...' cannot be found or is not useable."

How can I use commons-logging in an OSGi environment?

Commons-logging was not designed with OSGi in mind. This is why it is difficult to get commons-logging working in OSGi environments:

- LogFactory loads Log implementations by name (see Class.forName(String)). This is usually not possible in OSGi since every bundle classloader can only see the classes a bundle defines imports for.
- The bundle class loader that loads the commons-logging bundle will not have access to user provided commons-logging.properties files.
- commons-logging-api.jar contains classes that are also included in commons-logging.jar. This is contrary to traditional OSGi application architectures where one bundle defines an API and other bundles provide implementations for that API.

There alternatives to using commons-logging directly in OSGi are:

- Rebundled versions that contain proper OSGi meta data are available from Apache Felix, SpringSource and Eclipse Orbit.
- Using Pax logging.

Further information about this topic is available in the archives of the commons dev ML and the felix dev ML and in Jira.

Up to Logging