WSIF Trace

1 Quick summary

To switch WSIF trace on,in your log4j.properties specify the following settings to trace to wsif.log....

```
log4j.rootCategory=INFO, CONSOLE, LOGFILE

log4j.logger.org.apache.wsif.*=DEBUG
log4j.logger.com.ibm.wsif.*=DEBUG
log4j.appender.LOGFILE=org.apache.log4j.FileAppender
log4j.appender.LOGFILE.File=wsif.log
log4j.appender.LOGFILE.Append=true
log4j.appender.LOGFILE.Threshold=DEBUG
```

2 For those who want to know more

WSIF uses Apache commons-logging for its messages and trace. See http://jakarta.apache.org/commons/logging.html for more information. Commons-logging is an API which wraps various log implementations. You can configure commons-logging to use other log implementations, or even your own. So commons-logging.jar must be on the classpath, and log4j is the default log implementation, although commons-logging.jar contains other log implementations. You can configure commons-logging.properties to use a different log implementation. If using log4j, you configure log4j.properties to redirect trace and switch it on. The log4j.properties settings above are only a subset of all possibilities.

3 Interpreting trace

Here is a sample trace output...

```
7951 [main] DEBUG org.apache.wsif.*  -
ENTRY WSIFServiceFactory.newInstance()
8122 [main] DEBUG org.apache.wsif.*  -
EXIT  WSIFServiceFactory.newInstance(org.apache.wsif.base.WSIFServiceFactoryImpl@7df10a
8132 [main] DEBUG org.apache.wsif.*  -
ENTRY WSIFServiceFactoryImpl.getService<7df10a36>(C:\wsad-5\eclipse\workspace\ws-wsif\j
<null>, <null>, http://wsifservice.addressbook/, AddressBook)

8252 [main] DEBUG org.apache.wsif.*  -
ENTRY   PrivateCompositeExtensionRegistry.<init><43000a37>()
```

```
8893 [main] DEBUG org.apache.wsif.*  -
ENTRY    JavaBindingSerializer.registerSerializer<576c0a36>(com.ibm.wsdl.extensions.Pop
8913 [main] DEBUG org.apache.wsif.*  -
EXIT     JavaBindingSerializer.registerSerializer()
8943 [main] DEBUG org.apache.wsif.*  -
ENTRY    FormatBindingSerializer.<init><40b48a36>()

8943 [main] DEBUG org.apache.wsif.*  -
EXIT     FormatBindingSerializer.<init>()
9223 [main] DEBUG org.apache.wsif.*  -
ENTRY    FormatBindingSerializer.registerSerializer<40b48a36>(com.ibm.wsdl.extensions.P
9243 [main] DEBUG org.apache.wsif.*  -
EXIT     FormatBindingSerializer.registerSerializer()
9293 [main] DEBUG org.apache.wsif.*  -
ENTRY    EJBBindingSerializer.registerSerializer<55a24a36>(com.ibm.wsdl.extensions.Popu

9323 [main] DEBUG org.apache.wsif.*  -
EXIT     EJBBindingSerializer.registerSerializer()
9333 [main] DEBUG org.apache.wsif.*  -
EXIT    PrivateCompositeExtensionRegistry.<init>()
9353 [main] DEBUG org.apache.wsif.*  -
ENTRY   WSIFDynamicTypeMap.<init><32128a36>()
9353 [main] DEBUG org.apache.wsif.*  - EXIT    WSIFDynamicTypeMap.<init>()

9363 [main] DEBUG org.apache.wsif.*  -
ENTRY  WSIFServiceImpl.<init><31e38a36>(C:\wsad-5\eclipse\workspace\ws-wsif\java\sample
<null>, <null>, http://wsifservice.addressbook/, AddressBook)
9363 [main] DEBUG org.apache.wsif.*  - ENTRY   WSIFUtils.readWSDL(<null>,
C:\wsad-5\eclipse\workspace\ws-wsif\java\samples\addressbook\wsifservice\AddressBook.ws
9373 [main] DEBUG org.apache.wsif.*  -
ENTRY     WSIFPluggableProviders.getProvider(/)

9384 [main] DEBUG org.apache.wsif.*  -
ENTRY      WSIFPluggableProviders.getSupportingProviders(/, true)
9384 [main] DEBUG org.apache.wsif.*  -
ENTRY       WSIFPluggableProviders.getAllDynamicWSIFProviders()
9414 [main] DEBUG org.apache.wsif.*  -
ENTRY           WSIFPluggableProviders.readMETAINFClassNames(file:/C:/wsad-5/eclipse/wo
9424 [main] DEBUG org.apache.wsif.*  -
EVENT            WSIFPluggableProviders.readMETAINFClassNames Reading
provider class names from URL:
file:/C:/wsad-5/eclipse/workspace/ws-wsif/bin/META-INF/services/org.apache.wsif.spi.WSI
9434 [main] DEBUG org.apache.wsif.*  -
EVENT            WSIFPluggableProviders.readMETAINFClassNames Found
provider class name: org.apache.wsif.providers.ejb.WSIFDynamicProvider_EJB
9444 [main] DEBUG org.apache.wsif.*  -
EVENT            WSIFPluggableProviders.readMETAINFClassNames Found
provider class name:
org.apache.wsif.providers.java.WSIFDynamicProvider_Java
```

The columns before the ENTRY/EXIT are configurable in log4j.properties. In the sample above, the [main] shows that all these trace statements were made from the main thread. In a multithreaded application, the trace statements from all threads are interleaved. After the

ENTRY/EXIT is the WSIF classname.methodname indented according to stack depth. The hex number in angle brackets (<>) after the method name is the java object id, so it is possible to tell which object this method was run against. Methods which do not have an object id are static methods. After the object id are the parameters passed or returned from that method. Null parameters are represented as <null>. Some WSDL objects are represented as their fully qualified name and their object id. For instance definition({http://mynamespace}MyDefinition,1128e5e0). Some parameters may get traced over multiple lines.

Occasionally a method will be indented by two (or more) spaces than the method that called it, according to the trace. This is demonstrated in the sample trace above, by readMETAINFClassNames being indented by 5 more spaces than getAllDynamicProviders. This is because getAllDynamicProviders calls other private methods which aren't traced, which in turn call readMETAINFClassNames. So the indentation is a true reflection of the WSIF stack depth, but not all private methods get traced.

ENTRY trace statement represents a call to a method. An EXIT statement represents the return from a method. An EVENT statement represents other interesting information which may prove useful when diagnosing problems. An EXCEPTION trace statement represents a java exception at the moment that it is caught by WSIF. An ignored exception also represents a java exception at the moment that it caught by WSIF. The difference between an EXCEPTION and an ignored exception is that an EXCEPTION represents an unexpected problem (in the application, the WSDL, in WSIF or elsewhere), whereas an ignored exception represents an exception that was expected to be thrown and caught routinely as part of WSIF mainline code and does not represent a problem in itself. EXCEPTIONs are accompanied by their stack trace.

WSIF trace statements can be made from classes that are not part of WSIF. Such trace statements have their classname prefixed with their fully qualified package name. Commons-logging supports package-level tracing. That enables trace to be switched on or off for individual packages. This is not supported by WSIF. The only exception to this is tracing of WSIF logging itself. Tracing org.apache.wsif.logging.* enables trace to trace itself. This should be used with caution since traces produced this way are huge and difficult to use to diagnose problems that aren't in trace itself. A trace statement that contains "**** Exception in WSIF trace statement *******" represents a trace statement that itself has a bug in it. Such bad trace statements should not affect the normal operation of WSIF, whether or not trace is on.