Java CAPS 5.1.3 Notes
Logging

Michael Czapski, January 2008

Logging to additional destinations

Introduction

By default, the Sun SeeBeyond Integration Server domain uses a single log to log all events, whether arising out of execution of the Integration Server or arising out of execution of Java CAPS solutions deployed to it. This log is named `server.log` and is kept in the `<JavaVACPInstallRoot>/logicalhost/is/domains/<domainroot>/logs` directory.

This note discusses how the Integration Server can be configured to deliver all or selected events to an additional log handler and how XML-formatted event information can be obtained. This note also presents some ideas on multiple logging destinations and notification.

> ***Note***
> The writing of this note, and the underlying development and configuration work, were prompted by repeated questions about logging in Java CAPS, in particular how it could be made to behave like it used to in the non-J2EE versions of the SeeBeyond EAI products. For these readers who are still waiting for the answer they like – there will be no such answer coming. It is not possible to configure logging for 'your interface' separately from logging for all other 'interfaces' since there is no notion of 'your interface' as a separate component in Java CAPS. There is no way to collect logging information pertaining to a specific project or deployment in such a way that all relevant information is collected, no information is lost and no information on unrelated components is included.

Default logging configuration

Starting with release 5.1.0, Java CAPS comes with a Sun SeeBeyond Integration Server, which is a patched and modified version of the Sun Application Server 8.0 Platform Edition.

The default runtime platform for the Integration Server is Java 5 Standard Edition, or JDK 1.5. Java 5 Standard Edition provides logging support through the Java Logging APIs. Typically, logging in Java 5 SE is configured through the `logging.properties` configuration file in the platform's `jre/lib` directory. In a default Sun SeeBeyond Integration Server domain this configuration file is largely ignored.

Java Logging configuration and APIs documentation is readily available on the Internet so only issues pertinent to this note will be discussed. Readers with interest in this matter should consult the Java™ Logging Overview (http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/overview.html) and the Java

Logging APIs JavaDocs
(http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/overview.html).

Since the Application Server 8.0 uses the Java Logging APIs for logging so too does Java CAPS. By default, each Java CAPS domain logs to a `server.log` in the `<JavaVACPInstallRoot>/logicalhost/is/domains/<domainroot>/logs` directory.

The Integration Server overrides and supplements logging configuration provided in the platform's `logging.properties` configuration file. In particular, logging levels and logger categories are configured through the Integration Server Administration Console and stored in the `domain.xml` Integration Server's domain configuration file.

Some of the resulting Integration Server logging behaviour can still be modified through the `logging.properties` configuration file. This will be discussed later in this note.

Configuring the IS to use an additional handler

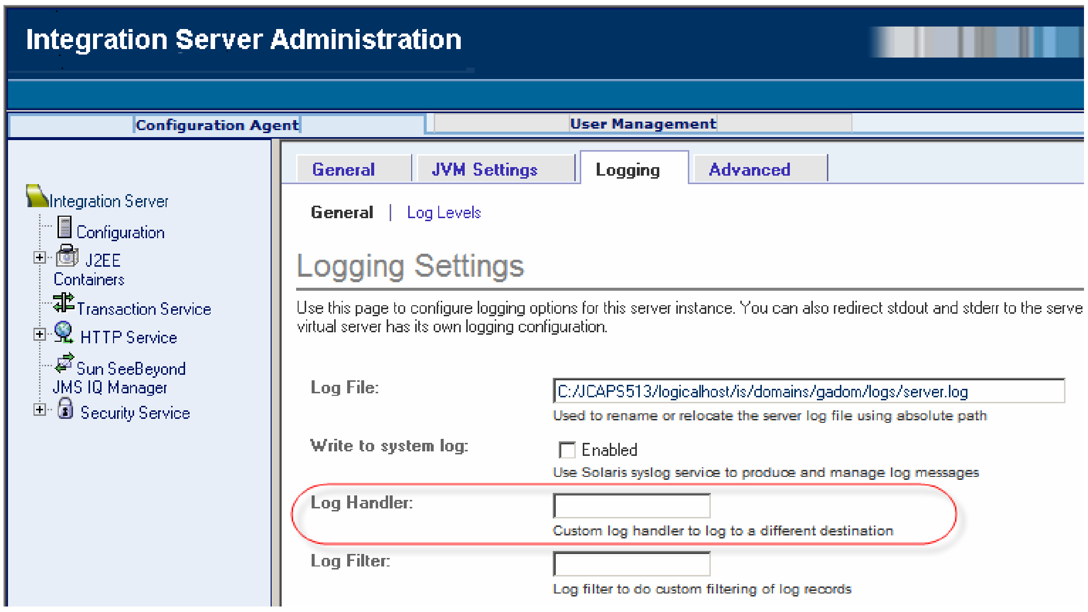
In Java Logging API a "Handler" is a Java class that receives logging records and does something with them. There are a set of Standard "handlers" supplied with the platform. These are:

- *StreamHandler:* A simple handler for writing formatted records to an OutputStream.
- *ConsoleHandler:* A simple handler for writing formatted records to System.err
- *FileHandler:* A handler that writes formatted log records either to a single file, or to a set of rotating log files.
- *SocketHandler:* A handler that writes formatted log records to remote TCP ports.
- *MemoryHandler:* A handler that buffers log records in memory.

The Java Logging API allows the user to specify multiple "handlers" so that logging records can be handled in different ways. Users can develop their own handlers that comply with the Java Logging APIs.

Notionally, one could specify multiple handlers in the `logging.properties` file. This would appear to work, with the globality caveats mentioned below, however in actuality only the Integration Server logging would go to the handlers so specified. No Java CAPS solutions log entries would go there. I don't know why this would be. Perhaps the IS overrides the handler chain or adds handlers after the `logging.properties` handlers are configured. In short, the 'interesting' logging records, these pertaining to Java CAPS solutions, will not be sent to handlers specified in the `logging.properties` configuration file.

The Sun SeeBeyond Integration Server is configured with a variant of a FileHandler that writes to the `server.log`. The Integration Server has a prevision for specifying one additional handler through the Integration Server Administration console's Logging Tab.



Logging Settings

Use this page to configure logging options for this server instance. You can also redirect stdout and stderr to the serve virtual server has its own logging configuration.

Log File:	C:/JCAPS513/logicalhost/is/domains/gadom/logs/server.log Used to rename or relocate the server log file using absolute path
Write to system log:	☐ Enabled Use Solaris syslog service to produce and manage log messages
Log Handler:	 Custom log handler to log to a different destination
Log Filter:	 Log filter to do custom filtering of log records

Unfortunately, there is no provision in the IS Administration console, or the `domain.xml`, for specifying configuration properties for this handler. This means that, if for example a standard FileHandler handler was specified, there would be no way to override the default name of the log file, default location to which it would be written, default format to use for log records and other properties that a FileHandler uses to modify its operation.

Fortunately, whilst it should not be used to add a handler, the `jre/lib/logging.properties` file for the logical host can be used to supply property values for the properties the handler specified through the IS Logging Tab uses to modify its runtime behaviour.

Unfortunately, the `jre/lib/logging.properties` file is global so that all Java CAPS IS domains under that logical host use the one file.

Unfortunately, the Sun SeeBeyond Integration Server ignores some of the configuration properties, for example the formatter, so only a limited degree of control over the additional handler, directly specified to the IS through the IS Administration console Logging Tab, is available through the configuration properties in the configuration file.

Fortunately, one can specify a MemoryHandler instead of another Handler and then configure the MemoryHandler to pass log records to that other Handler. The other Handler can then use all configuration file properties that pertain to it. This gives a handler indirection that overcomes the issue with inability to specify a formatter for directly configured handler, for example.

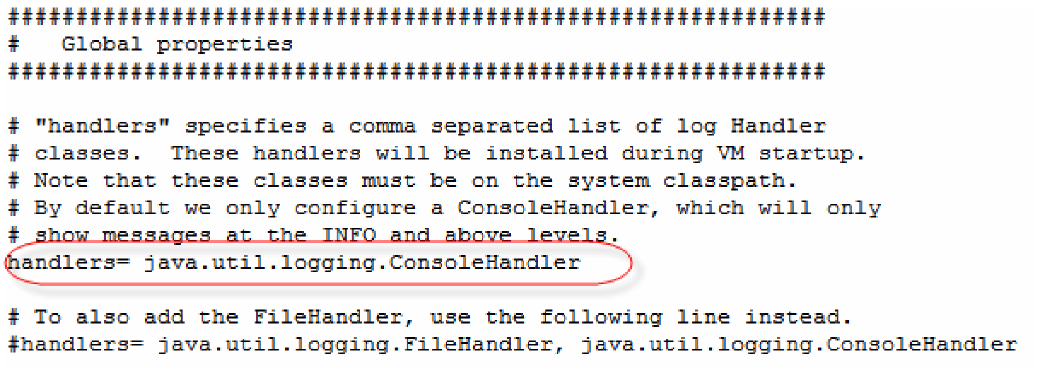
Preliminaries

Discussion below assumes that there is a Java CAPS 5.1.3 domain created and running. This domain will be started and stopped lots of times. It has a series of projects deployed and ready to exercise logging, most notably, projects discussed in the blog entry Java CAPS 5.1.3 - JCD-based Programmatic log manipulation (http://blogs.sun.com/javacapsfieldtech/entry/java_caps_5_1_311).

Adding a FileHandler through the logging.properties

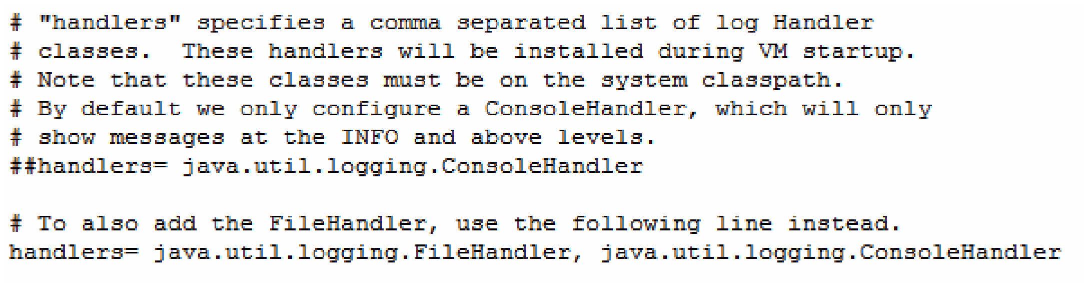
Let’s add a FileHandler to the jre/lib/logging.properties under the logicalhost – `<JCAPSInstallRoot>/logicalhost/jre/lib/logging.properties`.

The original file has the following handlers listed



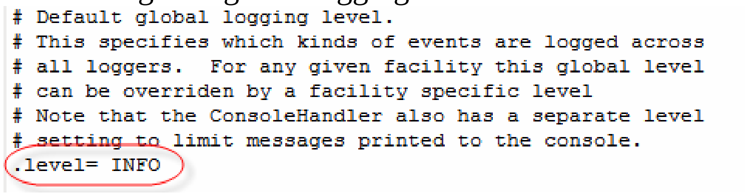
```
############################################################
#   Global properties
############################################################

# "handlers" specifies a comma separated list of log Handler
# classes.  These handlers will be installed during VM startup.
# Note that these classes must be on the system classpath.
# By default we only configure a ConsoleHandler, which will only
# show messages at the INFO and above levels.
handlers= java.util.logging.ConsoleHandler

# To also add the FileHandler, use the following line instead.
#handlers= java.util.logging.FileHandler, java.util.logging.ConsoleHandler
```

Commenting the first handlers line and uncommenting the second handlers line will add the FileHandler.



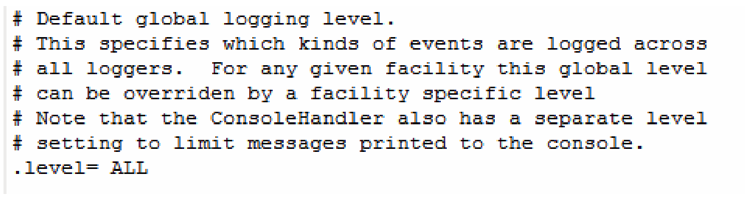
```
# "handlers" specifies a comma separated list of log Handler
# classes.  These handlers will be installed during VM startup.
# Note that these classes must be on the system classpath.
# By default we only configure a ConsoleHandler, which will only
# show messages at the INFO and above levels.
##handlers= java.util.logging.ConsoleHandler

# To also add the FileHandler, use the following line instead.
handlers= java.util.logging.FileHandler, java.util.logging.ConsoleHandler
```

Let’s change the global logging level from INFO



```
# Default global logging level.
# This specifies which kinds of events are logged across
# all loggers.  For any given facility this global level
# can be overriden by a facility specific level
# Note that the ConsoleHandler also has a separate level
# setting to limit messages printed to the console.
.level= INFO
```

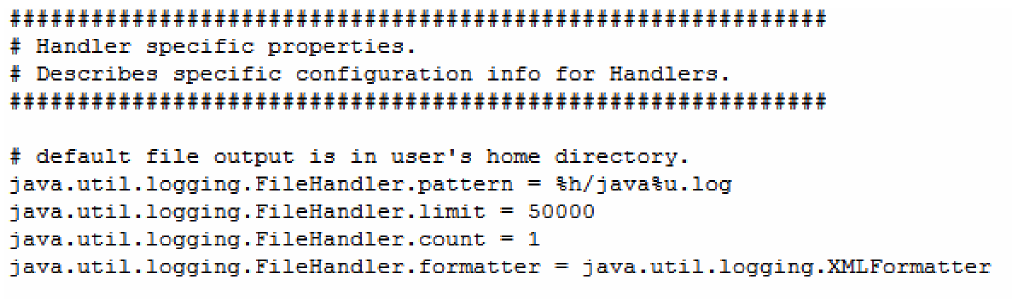
to ALL



```
# Default global logging level.
# This specifies which kinds of events are logged across
# all loggers.  For any given facility this global level
# can be overriden by a facility specific level
# Note that the ConsoleHandler also has a separate level
# setting to limit messages printed to the console.
.level= ALL
```

Let’s leave the rest of the original `logging.properties` file as is.

There are a series of properties pertaining to the FileHandler configuration. In the default `logging.properties` file created as part of Java CAPS installation these are:



```
############################################################
# Handler specific properties.
# Describes specific configuration info for Handlers.
############################################################

# default file output is in user's home directory.
java.util.logging.FileHandler.pattern = %h/java%u.log
java.util.logging.FileHandler.limit = 50000
java.util.logging.FileHandler.count = 1
java.util.logging.FileHandler.formatter = java.util.logging.XMLFormatter
```

See Java Doc for the java.util.logging.FileHandler (http://java.sun.com/j2se/1.5.0/docs/api/java/util/logging/FileHandler.html) class for definitions and explanations. In summary, with the settings as shown, the log file will be placed in the “user.home” directory, whatever that is for the particular platform, and will be called java<*nnnn*>.log, where <*nnn*> is a unique number. There will be at most one version of the file and it will be at most around 50K in size. It will contain XML-formatted log records.

Let’s re-start the domain, wait for it to go quiet and inspect the extra log file in the “user.home” directory. In this example the file will contain a series of records that look similar to the one shown below.

```
<record>
  <date>2008-01-11T12:10:15</date>
  <millis>1200013815828</millis>
  <sequence>1978</sequence>
  <logger>sun.rmi.client.ref</logger>
  <level>FINE</level>
  <class>sun.rmi.server.UnicastRef</class>
  <method>invoke</method>
  <thread>10</thread>
  <message>main: free connection (reuse = true)</message>
</record>
```

Because we have a global logging level set to ALL and we did not restrict the FileHandler there are a lot of logging records in the file. None of them, however, are identifiable as relating to out Java CAPS solutions. To make sure, let’s exercise the project ListLogCategories discussed in the blog mentioned before.

The regular Java CAPS log, the server.log, shows the following log entry (most omitted for brevity):

```
[#|2008-01-
11T12:15:08.906+1100|WARNING|IS5.1.3|STC.eGate.CMap.Collabs.ListLogCategories.cmListLo
gCategories_jcdListLogCategories1.LoggingListLogCategories.jcdListLogCategories|_Threa
dID=16; ThreadName=Worker: 3;
Context=project=ListLogCategories,deployment=dpListLogCategories,collab=cmListLogCateg
ories_jcdListLogCategories1,external=cListLogCategories;|
Loggers currently registered, with configured logging levels:

=INFO
```

```
ListLogCategories.cmListLogCategories_jcdListLogCategories1.jcdListLogCategories_Runti
me_Handler=INFO
STC.JMS.com.stc.jms.client.ConnectionImpl=INFO
STC.JMS.com.stc.jms.client.ObjectFactory=INFO
STC.JMS.com.stc.jms.client.QueueObjectFactory=INFO
STC.JMS.com.stc.jms.client.STCConnection=INFO
STC.JMS.com.stc.jms.client.STCConnectionFactory=INFO
STC.JMS.com.stc.jms.client.STCDestination=INFO
```

The additional log shown nothing – pretty useless. The IS fails to use the handler specified this way.

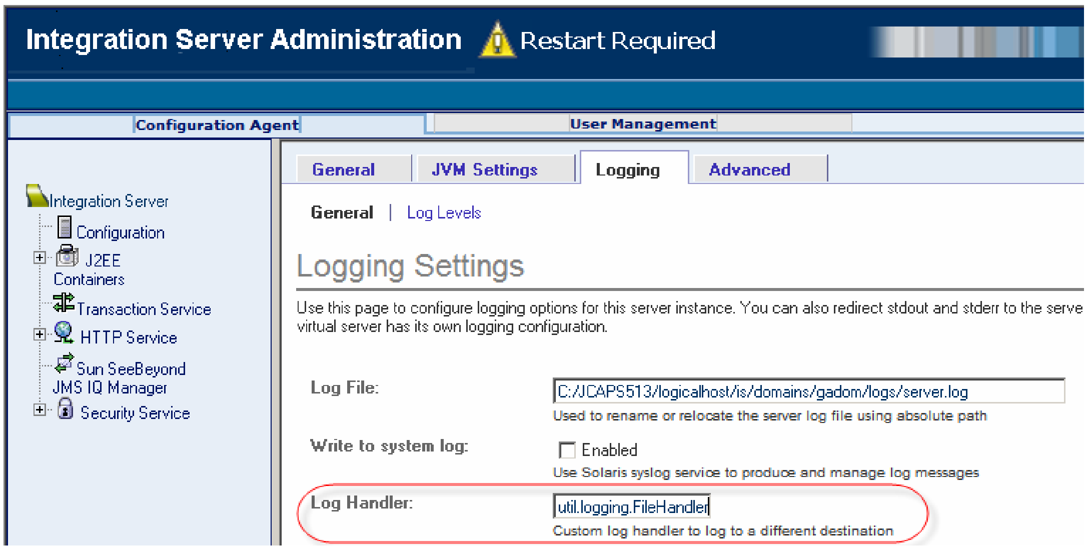
Let's comment out the handlers property with two handlers and uncomment the original handlers property for the next exercise. Let's leave the global logging level at ALL.

```
# show messages at the INFO and above levels.
handlers= java.util.logging.ConsoleHandler

# To also add the FileHandler, use the following line instead.
##handlers= java.util.logging.FileHandler, java.util.logging.ConsoleHandler

# Default global logging level.
# This specifies which kinds of events are logged across
# all loggers.  For any given facility this global level
# can be overriden by a facility specific level
# Note that the ConsoleHandler also has a separate level
# setting to limit messages printed to the console.
.level= ALL
```

Adding extra FileHandler directly to the IS

`java.util.logging.FileHandler` class is available in the `rt.jar`. It can be used as an additional handler for the IS.

Let's enter the string `java.util.logging.FileHandler` in the Logging Handler text box in the Integration Server Administration Logging Tab and press the "Save" button. Notice that the SI needs to be re-started.



Logging Settings

Use this page to configure logging options for this server instance. You can also redirect stdout and stderr to the serve
virtual server has its own logging configuration.

Log File: C:/JCAPS513/logicalhost/is/domains/gadom/logs/server.log
Used to rename or relocate the server log file using absolute path

Write to system log: ☐ Enabled
Use Solaris syslog service to produce and manage log messages

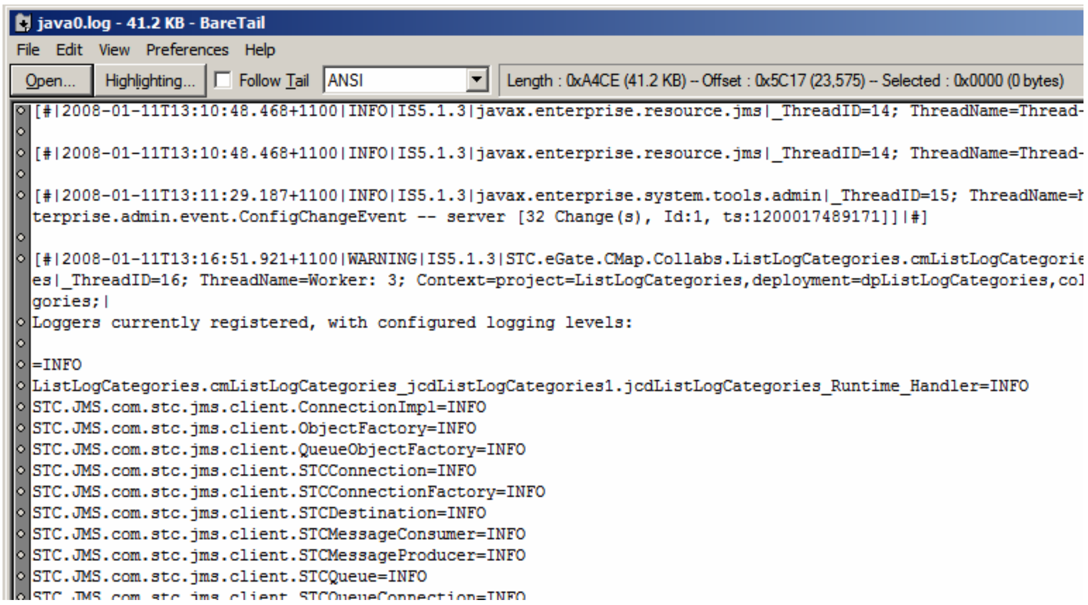
Log Handler: util.logging.FileHandler
Custom log handler to log to a different destination

The log file will be called java<*nnnn*>.log, as before, and will reside in the "user.home" directory, as before, since we have not changed relevant FileHandler properties in the `logging.properties` configuration file.

Let's re-start the IS and see what the additional log file shows.

Use a differences utility to see the differences between the java<nnn>.log and the server.log. The java<nnn>.log is a log smaller, because the FileHandler.limit property is set at 50000 bytes, however the last 50,000 bytes of both files are identical.

Let's exercise the ListLogCategories project. Where in the previous example none of the Java CAPS logging records went to the additional log, in this exercise, with the handler configured through the IS Administrator console, all records went to both logs.


java0.log - 41.2 KB - BareTail
File
Edit
View
Preferences
Help
Open...
Highlighting...
Follow Tail
ANSI
Length : 0xA4CE (41.2 KB) -- Offset : 0x5C17 (23,575) -- Selected : 0x0000 (0 bytes)

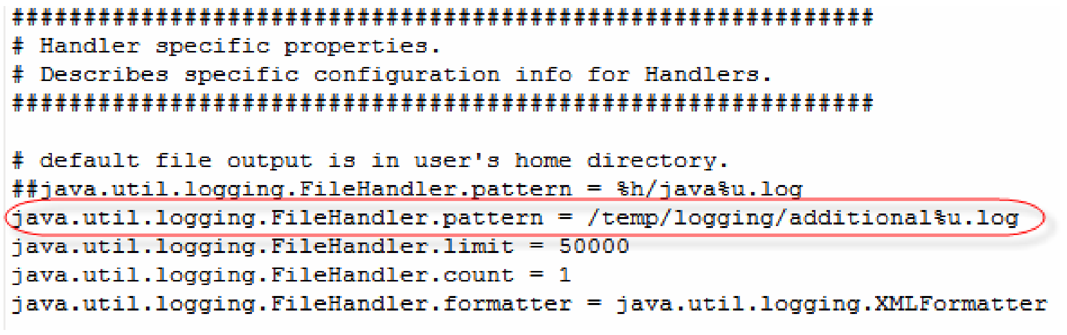
```
[#|2008-01-11T13:10:48.468+1100|INFO|IS5.1.3|javax.enterprise.resource.jms|_ThreadID=14; ThreadName=Thread-

[#|2008-01-11T13:10:48.468+1100|INFO|IS5.1.3|javax.enterprise.resource.jms|_ThreadID=14; ThreadName=Thread-

[#|2008-01-11T13:11:29.187+1100|INFO|IS5.1.3|javax.enterprise.system.tools.admin|_ThreadID=15; ThreadName=h
terprise.admin.event.ConfigChangeEvent -- server [32 Change(s), Id:1, ts:1200017489171]]|#]

[#|2008-01-11T13:16:51.921+1100|WARNING|IS5.1.3|STC.eGate.CMap.Collabs.ListLogCategories.cmListLogCategorie
es|_ThreadID=16; ThreadName=Worker: 3; Context=project=ListLogCategories,deployment=dpListLogCategories,col
gories;|
Loggers currently registered, with configured logging levels:

=INFO
ListLogCategories.cmListLogCategories_jcdListLogCategories1.jcdListLogCategories_Runtime_Handler=INFO
STC.JMS.com.stc.jms.client.ConnectionImpl=INFO
STC.JMS.com.stc.jms.client.ObjectFactory=INFO
STC.JMS.com.stc.jms.client.QueueObjectFactory=INFO
STC.JMS.com.stc.jms.client.STCConnection=INFO
STC.JMS.com.stc.jms.client.STCConnectionFactory=INFO
STC.JMS.com.stc.jms.client.STCDestination=INFO
STC.JMS.com.stc.jms.client.STCMessageConsumer=INFO
STC.JMS.com.stc.jms.client.STCMessageProducer=INFO
STC.JMS.com.stc.jms.client.STCQueue=INFO
STC.JMS.com.stc.jms.client.STCQueueConnection=INFO
```

Notice that the java<nnn>.log file no longer contains XML records. The FileHandler.formatter property in the logging.properties file, which was used to configure the formatter in the previous example, was completely ignored.

Changing FileHandler's Log File Name and Location

Let's modify `logging.properties` configuration file and specify different file name and location for the additional log file.

Let's set the file name to `/temp/logging/additional<nnn>.log` and re-start the IS.

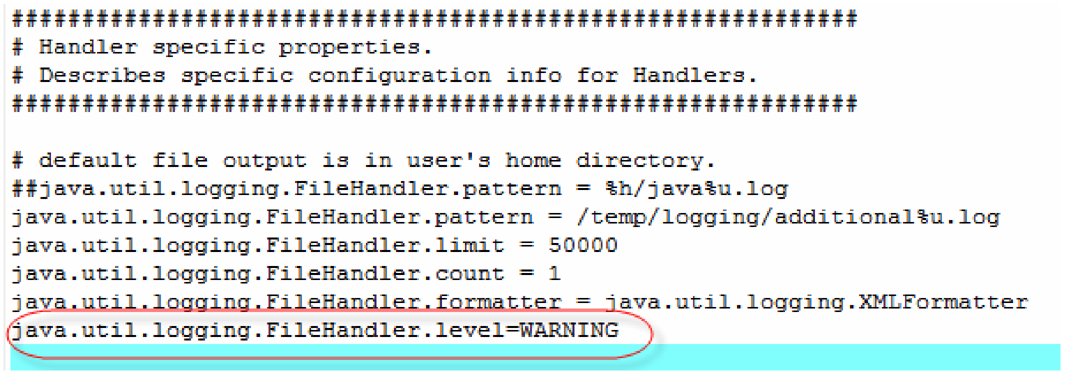

```
############################################################
# Handler specific properties.
# Describes specific configuration info for Handlers.
############################################################

# default file output is in user's home directory.
##java.util.logging.FileHandler.pattern = %h/java%u.log
java.util.logging.FileHandler.pattern = /temp/logging/additional%u.log
java.util.logging.FileHandler.limit = 50000
java.util.logging.FileHandler.count = 1
java.util.logging.FileHandler.formatter = java.util.logging.XMLFormatter
```

Obligingly, the new log file with the name of additional0.log was created in the specified directory, still in the same format as the `server.log` and still logging exactly the same information.

Filtering messages by Level

Because the global logging level in the `logging.properties` configuration file was set to ALL and no specific logging level was set for the FileHandler the log contained all the entries that the `server.log` contained.

We can restrict the entries that are logged to the additional log by specifying the FileHandler.level property and setting it to the appropriate value. Let's add the property and set it to the WARNING Level. The expectation is that only WARNING and more severe records will be logged to the additional log regardless of what gets logged to the `server.log`.


```
############################################################
# Handler specific properties.
# Describes specific configuration info for Handlers.
############################################################

# default file output is in user's home directory.
##java.util.logging.FileHandler.pattern = %h/java%u.log
java.util.logging.FileHandler.pattern = /temp/logging/additional%u.log
java.util.logging.FileHandler.limit = 50000
java.util.logging.FileHandler.count = 1
java.util.logging.FileHandler.formatter = java.util.logging.XMLFormatter
java.util.logging.FileHandler.level=WARNING
```

Let's re-start the IS. Notice that the additional0.log is empty. No information was logged (assuming there were no warning when the IS was starting☺)

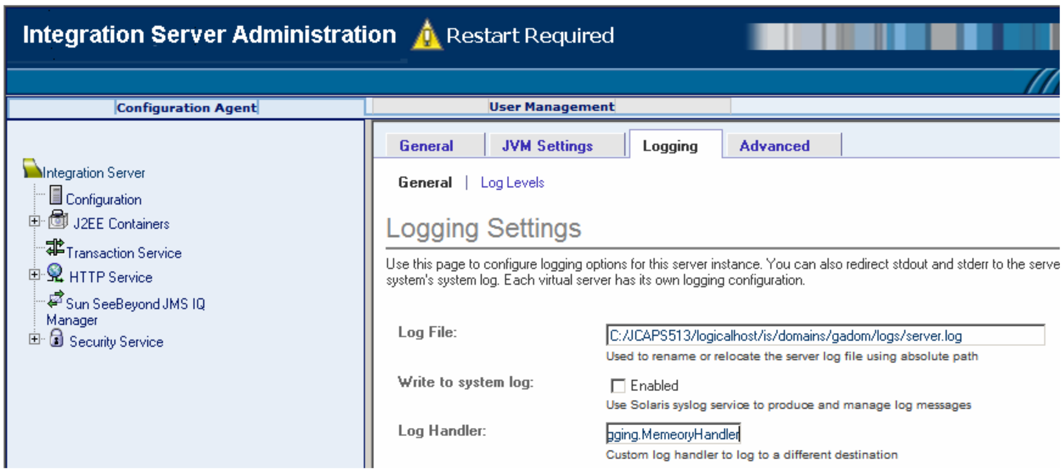
Let's exercise the `ListLogCategories` project. Notice that now a WARNING-level message was logged in the additional0.log. This is because the project emits a WARNING message.

With the `FileHandler.level` property set to WARNING, only WARNING and SEVERE messages will be logged to that log. With the property set to SEVERE only SEVERE level messages will get logged. This can be used as a simple filtering mechanism.

Getting XML format messages through the FileHandler

Since the IS somehow overrides the formatting property of the FileHandler we cannot use the FileHandler directly. A `java.util.logging.MemoryHandler` (`http://java.sun.com/j2se/1.5.0/docs/api/java/util/logging/MemoryHandler.html`) is a handler that logs to a circular buffer in memory and, when a log record of the "push level" or greater is logged, it pushes this log record to another handler, which it has been configured to use as the ultimate destination of log records.

Let's specifiy the java.util.logging.MemeoryHandler through the IS Administrator console Logging Tab as the additional handler to use instead of the FileHandler.



Logging Settings

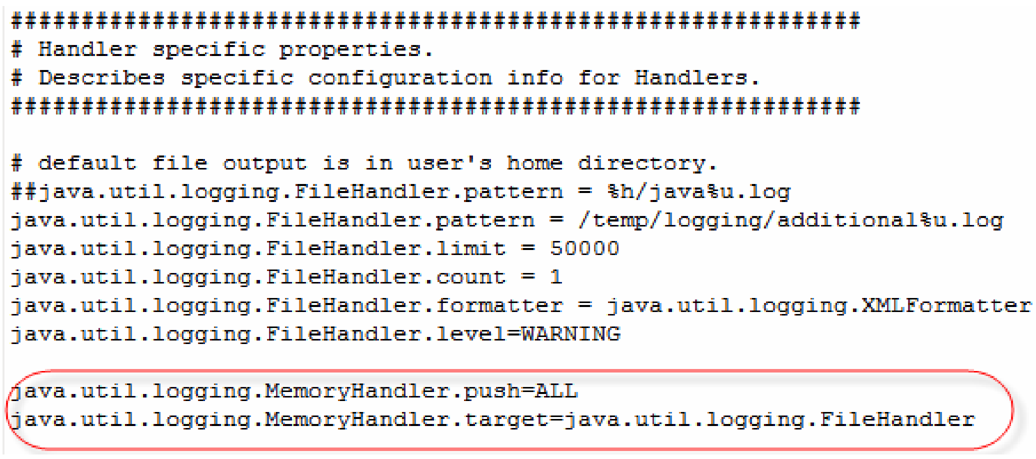
Use this page to configure logging options for this server instance. You can also redirect stdout and stderr to the serve system's system log. Each virtual server has its own logging configuration.

Log File: C:/JCAPS513/logicalhost/is/domains/gadom/logs/server.log
Used to rename or relocate the server log file using absolute path

Write to system log: ☐ Enabled
Use Solaris syslog service to produce and manage log messages

Log Handler: gging.MemeoryHandler
Custom log handler to log to a different destination

Before re-starting the IS let's configure the MemoryHandler through the logging.properties configuration file to use the FileHandler we used before as the ultimate destination of log reords.

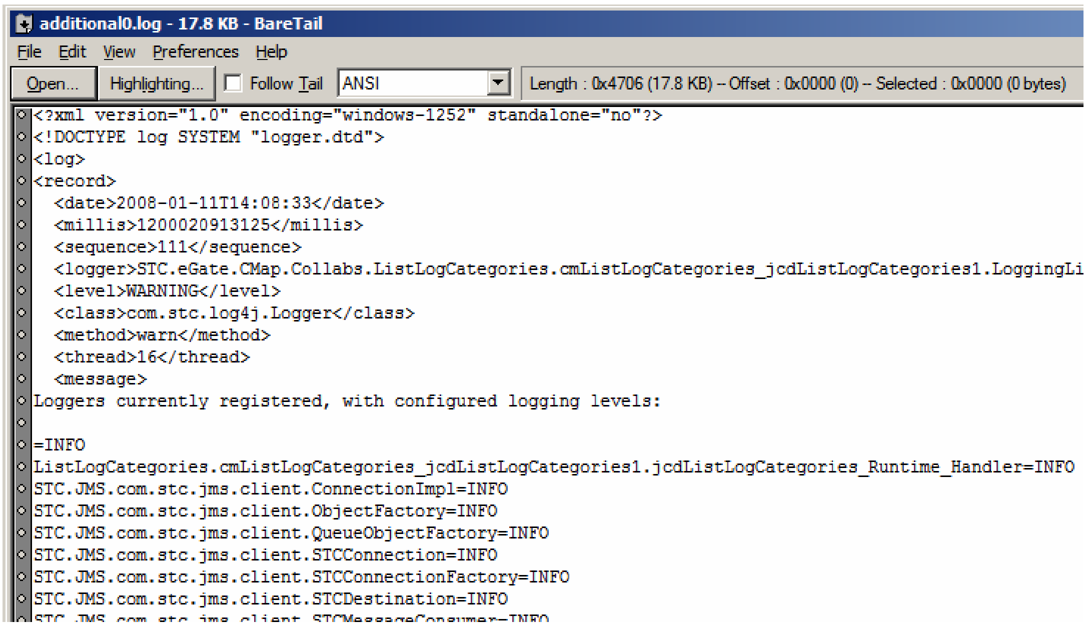


```
############################################################
# Handler specific properties.
# Describes specific configuration info for Handlers.
############################################################

# default file output is in user's home directory.
##java.util.logging.FileHandler.pattern = %h/java%u.log
java.util.logging.FileHandler.pattern = /temp/logging/additional%u.log
java.util.logging.FileHandler.limit = 50000
java.util.logging.FileHandler.count = 1
java.util.logging.FileHandler.formatter = java.util.logging.XMLFormatter
java.util.logging.FileHandler.level=WARNING

java.util.logging.MemoryHandler.push=ALL
java.util.logging.MemoryHandler.target=java.util.logging.FileHandler
```

The ultimate handler is the FileHandler. We expect all log records to be pushed to the FileHandler and be filtered by it to WARNING or more sever, as before.

Let's re-start the IS and observe the additional0.log log file configured to receive log records from the MemoryHandler. As in previous section, if no WARNING messages were logged during IS startup the additional log will be empty.

Let’s exercise the `ListLogCategories` project. Notice that now a WARNING-level message was logged in the additional0.log. This is because the project emits a WARNING message. Notice, too, that the log record in the additional0.log is in XML format. The FileHandler.formatter property setting has been recognised and used.


additional0.log - 17.8 KB - BareTail
File Edit View Preferences Help
Open... Highlighting... Follow Tail ANSI
Length : 0x4706 (17.8 KB) -- Offset : 0x0000 (0) -- Selected : 0x0000 (0 bytes)

```
<?xml version="1.0" encoding="windows-1252" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date>2008-01-11T14:08:33</date>
  <millis>1200020913125</millis>
  <sequence>111</sequence>
  <logger>STC.eGate.CMap.Collabs.ListLogCategories.cmListLogCategories_jcdListLogCategories1.LoggingLi
  <level>WARNING</level>
  <class>com.stc.log4j.Logger</class>
  <method>warn</method>
  <thread>16</thread>
  <message>
Loggers currently registered, with configured logging levels:

=INFO
ListLogCategories.cmListLogCategories_jcdListLogCategories1.jcdListLogCategories_Runtime_Handler=INFO
STC.JMS.com.stc.jms.client.ConnectionImpl=INFO
STC.JMS.com.stc.jms.client.ObjectFactory=INFO
STC.JMS.com.stc.jms.client.QueueObjectFactory=INFO
STC.JMS.com.stc.jms.client.STCConnection=INFO
STC.JMS.com.stc.jms.client.STCConnectionFactory=INFO
STC.JMS.com.stc.jms.client.STCDestination=INFO
STC.JMS.com.stc.jms.client.STCMessageConsumer=INFO
```

Different Formatter class can be developed and specified if appropriate. By default, an XMLFormatter is used if one not specified. Another standard formatter that is available in java.util.logging is the SimpleFormatter, which formats log records as shown in the server.log.

Discussion about development of Formatters is beyond the scope of this Note. See

Other Handlers

Using the MemoryHandler as the directly specified additional handler for the Sun SeeBeyond Integration Server, allows one to specify a different target Handler in such a way that all its properties are recognized and acted upon at runtime.

In addition to standard handlers supplied as part of the java.util.logging package a number of handlers have been developed by various people and are available for download. One of the ‘interesting’ ones is a SMTPHandler (http://smtphandler.sourceforge.net/) – a handler which sends log records as electronic mail.

Other handlers I have come across but not used include:

RollingFileHandler	http://www.x4juli.de/api/org/x4juli/handlers/RollingFileHandler.html
GenericHandler	http://monolog.objectweb.org/jdoc/ow_util_log/org/objectweb/util/monolog/wrapper/javaLog/GenericHandler.html
TableHandler	http://www-ui.is.s.u-

	tokyo.ac.jp/~kobayash/misc/doc/jp/ac/u_tokyo/s/is/www_ui/tomlib/logging/TableHandler.html
LogHandler	http://www.mit.edu/~vona/VonaUtils/javadoc-Vona/vona/log/Log.LogHandler.html
Java Logging YMSG Instant Message Handler and Formatter	http://www.planetsaturn.pwp.blueyonder.co.uk/ymsghandler/index.htm

Other handlers can be written as required.

I have come across a couple of articles that some might find interesting.

"Building the Ultimate Logging Solution"	http://java.sys-con.com/read/44698.htm
"Creating a JDBC Log Handler for JDK 1.4"	http://www.developer.com/java/article.php/1468351
"Using Log4j appenders with JDK logging"	http://www.kromosoft.com/resources/chainsawHandler/UsingChainsawWithJDKLogging.htm
"Gooey Logger"	https://aptframework.dev.java.net/gooey/logger.html

Using a SMTPHandler

Let's download the SMTPHandler from SourceForge (http://sourceforge.net/projects/smtphandler/), unzip to a convenient directory and copy the smtphandler-0.6.jar archive to the `logicalhost/is/lib/endorsed` directory. We could put it elsewhere but the archive must be in the IS's classpath.

In addition a JavaMail (http://java.sun.com/products/javamail/downloads/index.html) and Java Activation Framework (http://java.sun.com/products/javabeans/jaf/downloads/index.html) jars are required. Let's download them from their appropriate sites. From the JavaMail distribution extract the `mailapi.jar` and `smtp.jar` and copy them to the `logicalhost/is/lib/endorsed` directory. From the JAF distribution extract `activation.jar` and copy it to the `logicalhost/is/lib/endorsed` directory.

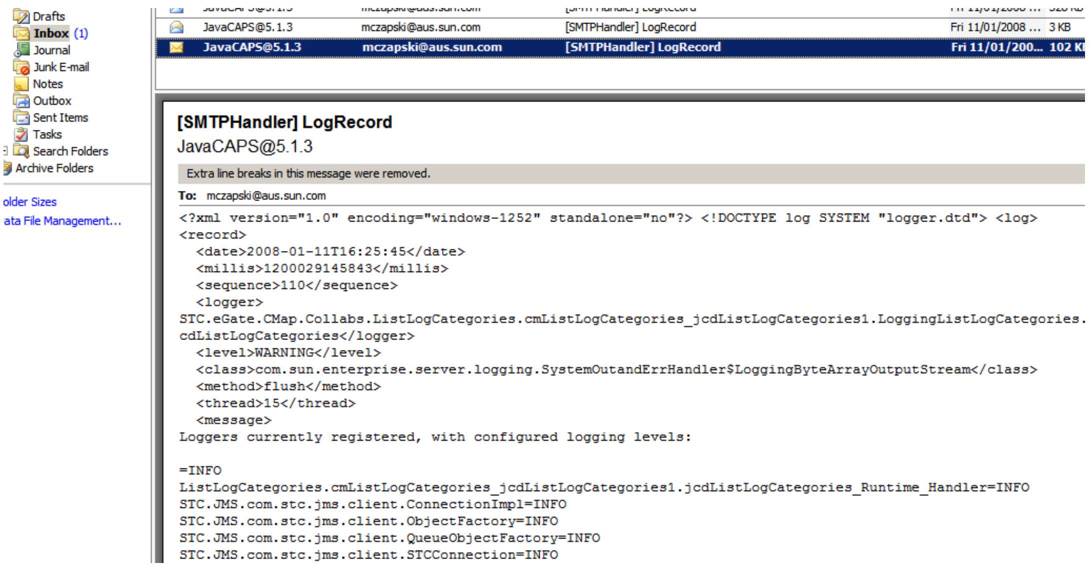
SMTPHandler seems to require permissions to read and write system properties. Let's add the following stanza to the end of the server.policy file.
Let's add SMTPHandler properties to the logging.properties configuration file and configure as appropriate for your SMTP server. Property values below are an example only.

```
# properties for SMTPHandler
smtphandler.SMTPHandler.level=WARNING
smtphandler.SMTPHandler.smtpHost=127.0.0.1
smtphandler.SMTPHandler.smtpUsername=mczapski@MyPostOffice
smtphandler.SMTPHandler.smtpPassword=mczapski
smtphandler.SMTPHandler.to=mczapski@aus.sun.com
smtphandler.SMTPHandler.from=JavaCAPS@5.1.3
smtphandler.SMTPHandler.subject=[SMTPHandler] LogRecord
smtphandler.SMTPHandler.bufferSize=100
smtphandler.SMTPHandler.formatter=java.util.logging.XMLFormatter
```

Let’s change the MemoryHandler.target property value to read:
```
java.util.logging.MemoryHandler.target=smtphandler.SMTPHandler
```

Let’s re-start the IS and exercise the `ListLogCategories` project.

If all is configured correctly, the Electronic Mail Client will receive a message similar to the following.



JavaCAPS@5.1.3	mczapski@aus.sun.com	[SMTPHandler] LogRecord	Fri 11/01/2008 ... 3 KB
JavaCAPS@5.1.3	mczapski@aus.sun.com	[SMTPHandler] LogRecord	Fri 11/01/200... 102 K

[SMTPHandler] LogRecord
JavaCAPS@5.1.3
Extra line breaks in this message were removed.
To: mczapski@aus.sun.com

```
<?xml version="1.0" encoding="windows-1252" standalone="no"?> <!DOCTYPE log SYSTEM "logger.dtd"> <log>
<record>
  <date>2008-01-11T16:25:45</date>
  <millis>1200029145843</millis>
  <sequence>110</sequence>
  <logger>
STC.eGate.CMap.Collabs.ListLogCategories.cmListLogCategories_jcdListLogCategories1.LoggingListLogCategories.
cdListLogCategories</logger>
  <level>WARNING</level>
  <class>com.sun.enterprise.server.logging.SystemOutandErrHandler$LoggingByteArrayOutputStream</class>
  <method>flush</method>
  <thread>15</thread>
  <message>
Loggers currently registered, with configured logging levels:

=INFO
ListLogCategories.cmListLogCategories_jcdListLogCategories1.jcdListLogCategories_Runtime_Handler=INFO
STC.JMS.com.stc.jms.client.ConnectionImpl=INFO
STC.JMS.com.stc.jms.client.ObjectFactory=INFO
STC.JMS.com.stc.jms.client.QueueObjectFactory=INFO
STC.JMS.com.stc.jms.client.STCConnection=INFO
```

All log records of severity WARNING and SEVERE will be sent, by electronic mail, to a configured mail recipient.

Some Ideas

With the ability to replicate logging information to an alternative destination all manner of possibilities open up to a Java programmers who don’t mind getting their hands a bit dirty.

The SMTPHandler used in the example points a way to using a Handler as a notification mechanism. With suitable handler one could send SEVERE-level message to a pager or a mobile telephone via a text message.

With a suitable handler one could send replicas of messages to multiple destinations. Modifying the MemoryHandler that comes with the java.util.logging could lead to a new MultiDestinationMemoryHandler that sends messages to multiple other handlers and implements independent configuration properties for each of the configured handlers.

Whereas filtering log records by specifying a logging level for a handler is useful but very coarse-grained, java.util.logging and the Sun SeeBeyond Integration Server support specification of a Filter class that determines which log records to pass on log. This may or may not become a subject of a separate note ☺

Other References

JavaTM Logging Overview -
http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/overview.html

JavaTM Logging APIs - http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/