IAM3: Websphere Message Broker – Node for log4j

User’s Guide

Version 2.1.1

May 2025

Matthias Nieder

IBM Deutschland

Property of IBM

Preface

Target audience

This SupportPac is designed for people who:
· Will be designing and implementing solutions using Websphere Message Broker.
· Are looking for alternatives to the Websphere Message Broker Trace node.
The reader should have a general awareness of the Websphere Message Broker and the Aapache log4j framework in order to make best use of this SupportPac.

The contents of this SupportPac

This SupportPac includes:
· the runtime component for using log4j in the Websphere Message Broker runtime.
· the Eclipse plug-in for extending the Websphere Message Broker Toolkit in order to develop flows that utilize the log4j logging system
· this User’s guide

Feedback on this SupportPac

We welcome constructive feedback on this SupportPac.
· Does it provide the features you expected ?
· Do you feel something important is missing?
· Is there too much technical detail, or not enough?
· Could the material be presented in a more useful manner?
Please direct any comments of this nature to **nieder@de.ibm.com**.

What’s new in version 2.1.1 ?

Fixed Bugs in version 2.1.1:
· fix bug to support IIB on z/OS
· fix bug to support ESQL functions

What’s new in version 2.1.0 ?

New functions in version 2.1.0:
· support log4j extensions via pluginFolder as policy (ACE) or Configurable Service (IIB)

What’s new in version 2.0.2 ?

Fixed Bugs in version 2.0.2:
· reenable configurable properties

What’s new in version 2.0.1 ?

Fixed Bugs in version 2.0.1:
· prevent NullPointerException if logLevel is DEBUG

What’s new in version 2.0.0 ?

New functions in version 2.0.0:
· IAM3 is now based on log4j version 2. log4j version 1.x is no longer supported
· Runtime node is packaged as a par-file
· Toolkit node appears in the palette under *Error handling*

What’s new in version 1.2.4 ?

Fixed Bugs in version 1.2.4:
· under heavy load there was a synchronization issue with loggers

What’s new in version 1.2.3 ?

Fixed Bugs in version 1.2.3:
· node fails to log ExceptionList in combination with RAW,RAWLF,XML,XMLLF

What’s new in version 1.2.2 ?

Fixed Bugs in version 1.2.2:
· ESQL initLog4j returned true, even if log4j initialization had failed

What’s new in version 1.2.1 ?

Fixed Bugs in version 1.2.1:
· node fails to log Environment and LocalEnvironment values

What’s new in version 1.2 ?

New functions in version 1.2:
· major performance improvements when handling larger messages
· LoggerName and LogLevel of a node instance can now be overridden by LocalEnvironment settings
· LocalEnvironment supported in LogText
· rework on the XMLLF output style

What’s new in version 1.1 ?

New functions in version 1.1:
· new XML XMLLF and RAWLF styles
· new XESQL XESQLF styles
· logger name is now configurable. If no special logger is defined the root logger is chosen (log4j standard)
· Properties, MQMD and other headers still need to be accessed using relative paths

Fixed Bugs in version 1.1:
· RAW style for whole environment now works with single slash (double slash still works, but should be changed)
· * in LogText is now allowed

Fixed Bugs in version 1.1.1:
· trying to log into different files using different logger names occasionally does not work correctly

Acknowledgements

I want to thank Tony Hays for his detailed descriptions of the default paths for the different broker versions and his installation instructions, which I found in the User's Guide of his SupportPac *IA91: WebSphere Message Broker – CacheNodes Broker Domain data store*.

I also want to thank Haiko Karg and Thomas Eppner for "Beta Testing" earlier versions of this SupportPac and giving valuable feedback to improve this User's Guide.

Table of Contents

Preface..2
Target audience..2
The contents of this SupportPac..2
Feedback on this SupportPac..2
What’s new in version 2.1.1 ?..2
What’s new in version 2.1.0 ?..2
What’s new in version 2.0.2 ?..2
What’s new in version 2.0.1 ?..2
What’s new in version 2.0.0 ?..3
What’s new in version 1.2.4 ?..3
What’s new in version 1.2.3 ?..3
What’s new in version 1.2.2 ?..3
What’s new in version 1.2.1 ?..3
What’s new in version 1.2 ?..3
What’s new in version 1.1 ?..3
Acknowledgements..4
1. Overview..7
2. Installation Instructions..8
2.1 Prerequisites..8
2.2 Broker installation..8
2.3 Toolkit installation..10
2.3.1 Toolkit installation for *App Connect Enterprise* 11.0.0.16 (and higher)..10
3. Migrating from earlier version..11
3.1 Prerequisites..11
3.2 Broker migration..11
3.3 Toolkit migration..12
3.4 Migrating your flows..12
3.4.1 Migrating the log4j nodes..12
3.4.2 Migrating the ESQL files..13
3.5 Toolkit update/upgrade..13
4. Using the log4j node..14
4.1 Node properties..14
4.2 LogText variables..15
4.2.1 XML message samples..16
4.2.2 BLOB message samples..19

5. Using log4j from ESQL....23
5.1 initLog4j function....23
5.2 log4j function....24
6. Known limitations....25
7. Troubleshooting....26
Appendix....27
A Sample log4j configuration file....27

1. Overview

Websphere Message Broker provides the Trace node for logging information to either the User Trace, the system log or to a file. But this node is kind of limited in the following ways:

- It is not extendable to other targets.
- There is no configurable log level.

The well known logging framework log4j from the Apache group solves exactly those limitations for Java based applications.

This SupportPac provides a node than can be used by your flows in order to address all targets reachable by log4j and configure log levels at runtime. For changing the log level you do not even need to restart your broker or flow. The change will be picked up by log4j after the amount of seconds that has been defined by the *monitorInterval* attribute in the log4j configuration file (e.g. monitorInterval="60").

From the log4j documentation:

“The ability to selectively enable or disable logging requests based on their logger is only part of the picture. Log4j allows logging requests to print to multiple destinations. In log4j speak, an output destination is called an *appender*. Currently, appenders exist for the console, files, GUI components, remote socket servers, JMS, SMTP and remote UNIX Syslog daemons. It is also possible to log asynchronously.”

For more information on log4j see:

https://logging.apache.org/log4j/2.x/manual/

log4j can be extended by writing your own appenders or extensions. For more information take a look at:

https://logging.apache.org/log4j/2.x/manual/extending.html

At the time of writing you can find a custom appender for SNMP under:

https://github.com/DushmanthaBandaranayake/log4j2-snmp-appender

There might exist other implementations and the link to the SNMP implementation is only a hint. It does not belong to IAM3 and there is no warranty or support from IAM3 for this implementation and its use.

2. Installation Instructions

This chapter describes the installation procedure for App Connect Enterprise version 11.0.0.16 or higher.

2.1 Prerequisites

You should have installed App Connect Enterprise V11.0.0.16 and App Connect Enterprise V11.0.0.16 or higher.

You need a JRE version 8 or higher configured to run your broker.

2.2 Broker installation

Some variables are used during the following instructions, because they vary from platform to platform:

App Connect Enterprise 11.0.0.16

Windows 10	
BROKER_INSTALL_DIR	C:\Program Files\IBM\ACE\11.0.0.16
MQSI_WORKPATH[1]	C:\ProgramData\IBM\MQSI
Linux, AIX, Solaris, HP-UX	
BROKER_INSTALL_DIR	/opt/IBM/ace-11.0.0.16
MQSI_WORKPATH[1]	/var/mqsi
z/OS	
BROKER_INSTALL_DIR	/usr/lpp/.....
MQSI_WORKPATH[1]	/var/mqsi

Table 1: Default variable values for the broker 11.0

App Connect Enterprise 12.0.3.0

Windows 10	
BROKER_INSTALL_DIR	C:\Program Files\IBM\ACE\12.0.3.0
MQSI_WORKPATH[1]	C:\ProgramData\IBM\MQSI
Linux, AIX, Solaris, HP-UX	
BROKER_INSTALL_DIR	/opt/IBM/ace-12.0.3.0
MQSI_WORKPATH[1]	/var/mqsi
z/OS	
BROKER_INSTALL_DIR	/usr/lpp/.....
MQSI_WORKPATH[1]	/var/mqsi

Table 1: Default variable values for the broker 12.0

[1] Each broker may specify a different MQSI_WORKPATH via mqsicreatebroker or mqsichange-broker.

Installation procedure:

1) Unzip the *iam3.zip* file and extract the following files:

   - Log4jLoggingNode_v2.1.1.par
   - ESQL/*

2) Copy the Log4jLoggingNode_v2.1.1.par file to *$BROKER_INSTALL_DIR/jplugin*.

3) If your flows make use of the ESQL functions:
   Copy all files from the ESQL folder to the $MQSI_WORKPATH/shared-classes for each broker.

4) Ensure that the broker has access to the files. For example, on Unix systems, you may want make these files group- and world-readable (via a chmod 755 command).

Optional:

If you have log4j extensions you can provide the necessary jar files in a custom folder. You must then make this folder available to the node by one of the following 2 methods:

1) IIB

   1. Define a Configurable Service in the WebUI of IIB with type “UserDefined” and name “IAM3”.
   2. Add a property with name “pluginFolder” and set your custom folder as the value.
      The value must always start with an “/”.

or

2) ACE

   1. Create a Policy Project in your toolkit with name “IAM3”.
   2. Create a policy in that project with name “IAM3” and type “UserDefined”
   3. Add a property “pluginFolder” and set your custom folder as the value.
      The value must always start with an “/”.
   4. Deploy the Policy Project to your integration server.

2.3 Toolkit installation

IAM3 is installed utilizing the feature concept of eclipse.

2.3.1 *Toolkit installation for App Connect Enterprise 11.0.0.16 (and higher)*

Installation procedure:

1) Unzip the *iam3.zip* file and extract the *Log4jLoggingPluginFeature_v2.0.2.zip* file.

2) Unzip this file into a installation directory of your choice, the so called *„update site“*.

3) Follow the instructions on how to install features or user-defined nodes using an update site:
   - Click on *Help → Install New Software...*
   - Click *Add* to choose your update site where you unzipped the file,
     click on *Local...* navigate to your update site and click *OK*
   - Select *Log4jLoggingPlugin,* (if it is not listed uncheck „*Group items by category*“)
     click on *Next*
   - Select *Log4jLoggingPlugin,* click on *Finish*

4) You need to restart your toolkit in order to make the new node show up in the palette (different to earlier versions the node is no longer under *Construction* but under *Error handling*, following up the new categories in ACE v12). If this does not work it might be necessary to restart the toolkit a second time or restart is with the –clean option.

3. Migrating from earlier version

This chapter describes the migration steps needed for App Connect Enterprise 11.0.0.16 or higher, if you have a previous version of this SupportPac installed.

In order to migrate from an earlier version it is recommended to migrate in the following order:

1) migrate the broker runtime, including replacing the log4j config file
2) migrate the toolkit
3) migrate your flows

If you are migrating from version >= 1.1 of this SupportPac you do not need to migrate the toolkit but by using the new version accept the new license and notices which can be found in the IAM3.zip file.

3.1 Prerequisites

You should have installed App Connect Enterprise V11.0.0.16 and App Connect Enterprise Toolkit V11.0.0.16 or higher.

3.2 Broker migration

For path variables see chapter *2.2 Broker installation*.

Migration procedure:

1) Stop Message Broker.

2) Unzip the *iam3.zip* file and extract the following files:

   - Log4jLoggingNode_v2.1.1.par
   - ESQL/*

Please notice that the dependent jar files (including log4j version 2.17.1) are now packaged inside the par-file.

3) Delete the file Log4jLoggingNode_vX.X.jar, any jakarta-oro and log4j jar-files (e.g. jakarta-oro-2.0.4.jar, log4j-1.2.8) from the directory *$MQSI_WORKPATH/shared-classes* for each broker and from the directory *$BROKER_INSTALL_DIR/jplugin (* where X.X is the version of SupportPac already installed (e.g. Log4jLoggingNode_v1.jar)).

4) Add the par-file to the *LILPATH* of the Broker or
   copy it to *$BROKER_INSTALL_DIR/jplugin*.

5) If your flows make use of the ESQL functions:
   Copy all files from the ESQL folder to the $MQSI_WORKPATH/shared-classes for each broker.

6) Replace the log4j configuration file with a log4j version 2 compatible configuration file (see appendix A Sample log4j configuration file)

7) Ensure that the broker has access to the files. For example, on Unix systems, you may want make these files group- and world-readable (via a chmod 755 command).

8) Start Message Broker.

3.3 Toolkit migration

For path variables see chapter *2.3 Toolkit installation*.

If you are migrating from version >= 1.1 of this SupportPac you do not need to migrate the toolkit but by using the new version you accept the new license and notices which can be found in the IAM3.zip file.

Migration procedure:

1) Stop the toolkit.

2) Delete the directory *$TOOLKIT_INSTALL_DIR/plugins/Log4jLoggingPlugin*
or
Uninstall the Log4jLoggingPlugin feature
(depending on your installation method)

3) Unzip the *iam3.zip* file and extract the *Log4jLoggingPluginFeature_v2.0.2.zip* file.

4) Follow the instructions for updating software utilizing the feature concept of eclipse.

5) You need to restart your toolkit in order to make the new node show up in the palette (different to earlier versions the node is no longer under *Construction* but under *Error handling*, following up the new categories in ACE v12). If this does not work it might be necessary to restart the toolkit a second time or restart is with the –clean option.

3.4 Migrating your flows

Because there is the new property *LoggerName* since version 1.1 you might also want to migrate your flows and ESQLs. This is an optional task. If nothing is changed, the new property is set to the default value “default”.

If you are migrating from version 1.1 of this SupportPac there is nothing to do concerning the flows !

3.4.1 Migrating the log4j nodes

There are two options to add the new property to the log4j nodes in your flow.

Migration using the MessageFlow editor:

For each instance of the log4j node in your flows:

1) open the properties view

2) overwrite the default entry “default” with the value you need.

Migration by editing the message flow source:

For each .msgflow file:

1) Search for each occurrence of the following enty:

```
<nodes xmi:type="Log4jLoggingPlugin.msgnode …… >
```

2) replace it with something like the following (e.g. for inserting “MyLogger” as the loggerName property entry):

```
<nodes xmi:type="Log4jLoggingPlugin.msgnode …… loggerName="MyLogger" >
```

3.4.2 Migrating the ESQL files

It is still possible to use the old style of the log4j() method.
In order to make use of the new logger name property you must add the logger name as the second parameter to your calls to log4j(). See section 5.2 for details.

If you are migrating from version >= 1.1 of this SupportPac there is nothing to do concerning ESQL !

3.5 Toolkit update/upgrade

If you encounter problems after upgrading the Message Broker Toolkit to a new version:

1) Look into $TOOLKIT_INSTALL_DIR\configuration\org.eclipse.update.
2) Save a version of the file platform.xml.
3) Edit the file platform.xml as follows:
   Replace the line:

   <site enabled="true" policy="USER-EXCLUDE" updateable="true" list="Log4jLoggingPlugin/" url="platform:/base/">

   with:

   <site enabled="true" policy="USER-EXCLUDE" updateable="true" url="platform:/base/">
4) Restart the toolkit with the -clean option.

4. Using the log4j node

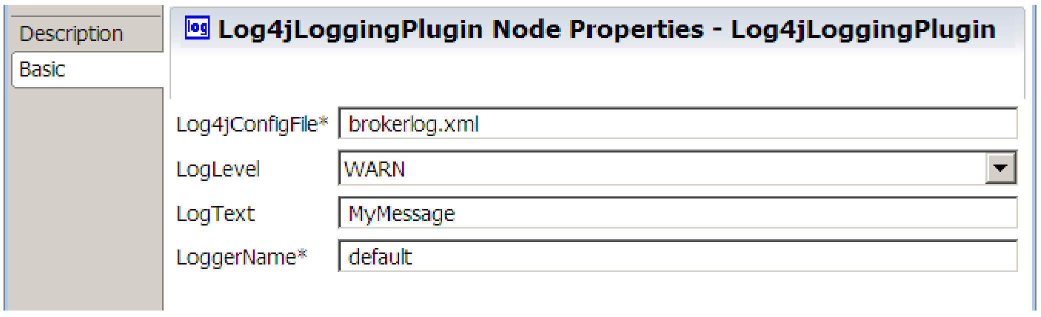
The icon for the log4j node is the following:

You find it in the message flow palette under the section *Construction*. The node has an *Input* and an *Output* terminal, so you can simply put it at any position in your flow. The message itself is not changed by the node.

The only exception to the latter is if you use ESQL in the *LogText* property (see below) that changes the message. It is not recommended to do that !

4.1 Node properties

The log4j node has the following properties:



Log4jLoggingPlugin Node Properties - Log4jLoggingPlugin

Log4jConfigFile*	brokerlog.xml
LogLevel	WARN
LogText	MyMessage
LoggerName*	default

Figure 1: Properties of the log4j node

***<u>Log4jConfigFile</u>*:**

This field specifies the name of the log4j configuration file at runtime. Make sure that the path to the file is in the broker's CLASSPATH. It need not be present in the toolkit. To be save it is recommended to specify an absolute path in a first step.
For solving problems see section *"Troubleshooting"*.

This field is mandatory.

Although this is a mandatory field it is recommended to use the same value in each occurrence of this node in your flow. See section "Known limitations" for details.

***<u>LogLevel</u>*:**

This field sets the level of this log text. At runtime only log texts with a level greater or equal to the level specified in the log4j configuration file (*broker.xml* in the sample above) will be logged.

You can override this setting by setting *LocalEnvironment.Log4j.LogLevel* (e.g. using ESQL in a ComputeNode before this node).

***LogText*:**

This value is the text you want to log and builds the log text. It can take variables as described in section "LogText variables".

This field is ignored if a there has been a value assigned to the following field before the log4j node has been reached:

Environment.Variables.Log4j.LogText

In this case the value of this field takes precedence. To indicate this, this value is also the default value for this field.

In addition the log4j node maps the flow name to the thread name property in log4j. This makes it possible to identify log output from different flows in a single log file.

***LoggerName*:**

This value is used to define a log4j logger for this statement. In log4j a logger can be used to differentiate inside the configuration. If no logger is defined, everything is passed to the default logger, which is called root or the root logger.

Appenders are mapped to one or more loggers. If you have appenders for writing to different files, you can use different loggers to map them to the different appenders/files in your log4j configuration.

You can override this setting by setting *LocalEnvironment.Log4j.LoggerName* (e.g. using ESQL in a ComputeNode before this node).

4.2 LogText variables

Variables in your *LogText* field are used to log dynamic context that is derived from parts of the message at runtime. All variables in your message begin and end with '$' character. There are different variable styles:

1) XPath style
2) RAW, RAWLF style
3) ESQL style
4) XML, XMLLF style
5) XESQL, XESQLLF style

You can mix static parts and variables of every style in a single log text as you like.

The "LF" styles are the same like the styles without LF with the difference that they put in line feeds ("LF") to make it more readable.

To understand the different styles take a look at the following examples.

4.2.1 XML message samples

For all examples in this section imagine the following message is flowing through the node:

MQHeader:	Format = MQSTR
	CCSID = 1252
MQData:	<Test>test<Part>PART</Part></Test>

The following examples assume that there is no MRM definition available. So the type of message in the examples is assumed to be XMLNS.

XPath:

Example 1: XPath with Root

LogText = MyMessage $/$

Result:

```
11:55:14,010 WARN  [default.Log4jXmlTestFlow] MyMessage is testPART
```

Remark:

Using XPath the log4j node always wraps the result of your expression with the XPath string() method.

With XPath the "/" maps to *Root.* So all elements of the message tree are tried to be displayed as a String.

Example 2: XPath with message name

LogText = My Message is $/Test$

Result:

```
11:55:14,010 WARN  [default.Log4jXmlTestFlow] MyMessage is testPART
```

Remark:

For XML messages there is no difference in the output between *Root* and the XML root tag (message name).

Example 3: XPath with a subpart

LogText = My Message is $/Test/Part$

Result:

```
11:55:14,010 WARN  [default.Log4jXmlTestFlow] MyMessage is PART
```

Remark:

Parts of a message can be accessed as usual.

RAW:

Example 4: RAW with Root

LogText = My Message is $RAW:/$

Result:

```
11:55:14,010 WARN  [default.Log4jXmlTestFlow] MyMessage is Root.MbElement( type: 1000000 name: XMLNS ), XMLNS.MbElement( type: 1000000 name: Test ) { test }, Test.MbElement( type: 1000000 name: Part ) { PART }
```

Remark:

The RAW style disables the string() wrapping of the XPath style and displays more detailed information.

Example 5: RAW with message name

LogText = My Message is $RAW:/Test$

Result:

```
11:55:14,010 WARN  [default.Log4jXmlTestFlow] MyMessage is XMLNS.MbElement( type: 1000000 name: Test ) { test }, Test.MbElement( type: 1000000 name: Part ) { PART }
```

Remark:

Using the message name with the RAW style only the *Root* element information is omitted.

ESQL:

Example 6: ESQL with message name

LogText = My Message is $ESQL:InputRoot.XMLNS.Test$

Result:

```
11:55:14,010 WARN  [default.Log4jXmlTestFlow] MyMessage is testPART
```

Remark:

As with XPath when using the ESQL style the log4j node always wraps the result of your expression with a XPath like string() method.

Using the ESQL style the field delimiter is “.” (not “/” as with XPath). You can access the message as you would with normal ESQL.

Example 7: ESQL with a subpart

LogText = My Message is $ESQL: InputRoot.XMLNS.Test.Part$

Result:

```
11:55:14,010 WARN  [default.Log4jXmlTestFlow] MyMessage is PART
```

Remark:

Subparts of the message can be accessed with the ESQL style as with normal ESQL.

XML:

Example 7: XML with message name

```
LogText = My Message is $XML:/Test$
```

Result:

```
11:55:14,010 WARN  [default.Log4jXmlTestFlow] MyMessage is <Test>test<Part>PART</Part></Test>
```

Remark:

The XML style prints out the given part of the message tree (if the message itself is XML or not) in an XML style.

Example 8: XML with a subpart

```
LogText = My Message is $XML:/Test/Part$
```

Result:

```
11:55:14,010 WARN  [default.Log4jXmlTestFlow] MyMessage is <Part>PART</Part>
```

Remark:

The root tag of a message part is the name of the subpart.

XESQL:

Example 9: XESQL with message name

```
LogText = My Message is $XESQL:InputRoot.XMLNS.Test$
```

Result:

```
11:55:14,010 WARN  [default.Log4jXmlTestFlow] MyMessage is <Value>test<Part>PART</Part></
Value>
```

Remark:

The XESQL style in a way combines the XML and ESQL styles. It uses the addressing part of the ESQL style and prints out the given part of the message tree (if the message itself is XML or not) in an XML style. The root tag of the output is always “Value”.

Example 10: XESQL with a subpart

```
LogText = My Message is $XESQL:InputRoot.XMLNS.Test.Part$
```

Result:

```
11:55:14,010 WARN  [default.Log4jXmlTestFlow] MyMessage is <Value>PART</Value>
```

Remark:

The root tag of the output is always “Value”.

4.2.2 BLOB message samples

For all examples in this section imagine the following message is flowing through the node:

MQHeader:	Format = MQSTR
	CCSID = 1252
MQData:	Test11

If you have specified a structure in the MRM you can access this message or parts of it as you would with XPath or ESQL as usual. The following examples assume that there is no MRM definition available. So the type of message in the examples is assumed to be BLOB.

XPath:

Example 1: XPath with Root

LogText = My Message is $/$

Result:

```
16:14:32,368 WARN  [default.Log4jTest] My Message is MQSTR546573743131
```

Remark:

Using XPath the log4j node always wraps the result of your expression with the XPath string() method.

With XPath the "/" maps to *Root.* So all elements of the message tree are tried to be displayed as a String. But only the format of the MQ header and the data as a hex string is displayed.

Example 2: XPath with a subpart

LogText = My Message is $BLOB/BLOB$

Result:

```
16:14:32,368 WARN  [default.Log4jTest] My Message is Test11
```

Remark:

Accessing the subpart of the BLOB message it is possible to display the data as you would like to see it in the log.

With XPath you can start either with

- "/", which maps to *Root*,
- the message type (BLOB in this sample),
- "Environment"
- "LocalEnvironment"

or you choose $ExceptionList$, which prints out an exception stack if there is one.

Example 3: XPath with Environment

LogText = An Environment value: $Environment/Variables/MyValue$!

Result:

```
16:14:32,368 WARN  [default.Log4jTest] An Environment value: Test11 !
```

Remark:

You must have set the field *Environment/Variables/MyValue* in the flow before the log4j node is reached !

RAW:

Example 4: RAW with Root

LogText = My Message is $RAW:/$

Result:

```
16:14:32,368 WARN  [default.Log4jTest] My Message is Root.MbElement( type: 1000000 name: BLOB ),
BLOB.MbElement( type: 3000000 name: UnknownParserName value: MQSTR ),
BLOB.MbElement( type: 3000000 name: BLOB value:[B@2d162d16 )
```

Remark:

The RAW style disables the string() wrapping of the XPath style.

ESQL:

Example 5: ESQL with subpart

LogText = My Message is $ESQL:InputRoot.BLOB.BLOB$

Result:

```
16:14:32,368 WARN  [default.Log4jTest] My Message is 546573743131
```

Remark:

As with XPath when using the ESQL style the log4j node always wraps the result of your expression with a XPath like string() method.

Using the ESQL style the field delimiter is “.” (not “/” as with XPath). Although BLOB.BLOB is specified, the text is still logged as a hex string because there is no wrapping with string() as it is with XPath.

Example 6: ESQL with embedded ESQL functions

LogText = My Message is $ESQL:CAST(InputRoot.BLOB.BLOB AS CHARACTER CCSID 1252)$

Result:

```
16:14:32,368 WARN  [default.Log4jTest] My Message is Test11
```

Remark:

The BLOB field is converted to a character using the ESQL CAST function. The result is the text as you would like to see it in the log.

XML:

Example 7: XML with Root

LogText = My Message is $XML:/$

Result:

```
16:14:32,368 WARN  [default.Log4jTest] My Message is <BLOB><UnknownParserName>MQSTR</UnknownParserName><BLOB>54 65 73 74 31 31 </BLOB></BLOB>
```

Remark:

The XML style prints out the given part of the message tree (if the message itself is XML or not) in an XML style.

Example 8: XML with a subpart

LogText = My Message is $XML:BLOB/BLOB$

Result:

```
16:14:32,368 WARN  [default.Log4jTest] My Message is <BLOB>54 65 73 74 31 31 </BLOB>
```

Remark:

Accessing the subpart of the BLOB message it is possible to display the data as you would like to see it in the log.

As with XPATH using XML you can start either with

- “/”, which maps to *Root*,
- the message type (BLOB in this sample),
- “Environment”
- “LocalEnvironment”

or you choose $XML:ExceptionList$, which prints out an exception stack if there is one.

XESQL:

Example 9: XESQL with subpart

LogText = My Message is $XESQL:InputRoot.BLOB.BLOB$

Result:

```
16:14:32,368 WARN  [default.Log4jTest] My Message is <Value>54 65 73 74 31 31 </Value>
```

Remark:

As with ESQL using the XESQL style the field delimiter is “.” (not “/” as with XPath). Although BLOB.BLOB is specified, the text is still logged as a hex string because there is no wrapping with string() as it is with XPath.

The XESQL style in a way combines the XML and ESQL styles. It uses the addressing part of the ESQL style and prints out the given part of the message tree (if the message itself is XML or not) in an XML style. The message (part) is always enclosed wit the tag “Value”.

Example 10: XESQL with embedded ESQL functions

LogText = My Message is $XESQL:CAST(InputRoot.BLOB.BLOB AS CHARACTER CCSID 1252)$

Result:

```
16:14:32,368 WARN  [default.Log4jTest] My Message is <Value>Test11</Value>
```

Remark:

The BLOB field is converted to a character using the ESQL CAST function. The result is the text as you would like to see it in the log.

The XESQL style always encloses the message (part) with the tag “Value”.

5. Using log4j from ESQL

This SupportPac also provides an API to use the log4j functionality from your ESQL code. This API is made up of three functions:

- initLog4j
- log4j
- log4j_1_1
- log4j_2

The second function (“log4j”) is to provide backward compatibility to version 1.0 of this SupportPac.
The third function is identical to the log4j_2 function and provided for backward compatibility to Version 1.1.x and version 1.2.x of this SupportPac. You can use both of them interchangeable depending on what you declare in your ESQL.
You need to declare these external functions in your Compute Node modules in order to use the log4j API in ESQL:

```
CREATE FUNCTION initLog4j( IN CONFIG_FILE_NAME CHARACTER )
      RETURNS BOOLEAN
      LANGUAGE JAVA
      EXTERNAL NAME "com.ibm.broker.IAM3.Log4jNode.initLog4j";


CREATE FUNCTION log4j( IN COMPONENT_NAME CHARACTER,
                       IN LEVEL CHARACTER,
                       IN TEXT CHARACTER )
      RETURNS BOOLEAN
      LANGUAGE JAVA
      EXTERNAL NAME "com.ibm.broker.IAM3.Log4jNode.log";


CREATE FUNCTION log4j_1_1( IN COMPONENT_NAME CHARACTER,
                       IN LOGGER_NAME CHARACTER,
                       IN LEVEL CHARACTER,
                       IN TEXT CHARACTER )
      RETURNS BOOLEAN
      LANGUAGE JAVA
      EXTERNAL NAME "com.ibm.broker.IAM3.Log4jNode.log";

CREATE FUNCTION log4j_2( IN COMPONENT_NAME CHARACTER,
                       IN LOGGER_NAME CHARACTER,
                       IN LEVEL CHARACTER,
                       IN TEXT CHARACTER )
      RETURNS BOOLEAN
      LANGUAGE JAVA
      EXTERNAL NAME "com.ibm.broker.IAM3.Log4jNode.log";
```

5.1 initLog4j function

The *initLog4j* function needs to be called only once in your flow. It initializes the log4j system by providing the log4j configuration file as a parameter:

```
DECLARE rc BOOLEAN;
CALL initLog4j('brokerlog.xml') INTO rc;
IF (  rc = FALSE ) THEN
-- error initializing log4j
…
END IF;
```

If you do not specify an absolute path to the file (*brokerlog.xml* in the example above) make sure that the path to the file is in the broker's CLASSPATH. To be save specify an absolute path in a first step. For solving problems see section *"Troubleshooting"*.

Subsequent calls to this method are without effect once the log4j system is initialized.

5.2 log4j function

The *log4j* method logs your text using log4j. It takes three parameters:

***COMPONENT_NAME*:**
This value is mapped to the thread name property in log4j. A suggestion is to always use the name of the flow in this parameter, because this is also the behaviour of the log4j node.

***LOGGER_NAME*:**
This value is mapped to the logger name property in log4j. If you use the old version without this parameter, the COMPONENT_NAME is also mapped to this property.

***LEVEL*:**
This value is the log level of this log text. At runtime only log texts with a level greater or equal to the level specified in the log4j configuration file will be logged.

***TEXT*:**
This value is the text you want to log and builds the log text. It is not possible to use any variables as described in section "LogText variables".The text should be built up in ESQL before.

```
DECLARE rc BOOLEAN;
CALL log4j('MyFlow', 'WARN','Message from ESQL') INTO rc;
IF ( rc = FALSE ) THEN
-- error during logging
…
CALL log4j_1_1('MyFlow', 'File1', 'WARN','Message from ESQL') INTO rc;
CALL log4j_2('MyFlow', 'File1', 'WARN','Message from ESQL') INTO rc;

END IF;
```

6. Known limitations

1) The property *Log4jConfigFile* is mandatory for the node, but after a successful initialization of the log4j logging system this field is ignored in further occurrences of the log4j node in the flow. Even more this field is ignored as long as the execution group (Java VM) is running.

   Since ESQL uses a different class loader the ESQL API initializes its own log4j system. Again, once the *initLog4j* function was successful subsequent calls to this function are without effect as long as the execution group (Java VM) is running.

2) The name of the flow is mapped to the thread property for log4j using the log4j node. In ESQL you must provide this name explicitly because it is not accessible in this context because of ESQL to Java mappings.

3) The XPath and ESQL style always wraps the result of an expression with the string() function and no other XPath function is supported. Take a look at the XML or XESQL styles if you want to output additional information.

4) The field *Environment.Variables.Log4j.LogText* takes precedence if it has a value when the log4j node is reached.

7. Troubleshooting

Where to find information:

In case of an internal error log4j writes some information to stdout.

For broker 6.1 on Windows this can be found at:

$MQSI_WORKPATH\components\<BrokerName>\<ID>\console.txt

For broker 6.1 on UNIX this can be found at:

$MQSI_WORKPATH\components\<BrokerName>\<ID>\stdout

$MQSI_WORKPATH\components\<BrokerName>\<ID>\stderr

For other operating systems refer to your product documentation. It might not be supported on Windows brokers version 6.0.

log4j configuration file not found:

If the log4j configuration file can not be found, make sure you have set up the CLASSPATH correctly. If you change the CLASSPATH you must restart the broker to make this change effective.

If you first run with a fully qualified path name and change this later you also must restart the broker, because once log4j is initialized it will ignore the configuration file settings of subsequent node occurrences (see Known limitations).

Time issues in the output:

The timestamps used for the log statements are computed by the Java VM. If you have issues with that, you must configure the following JVM start parameter for the broker JVM:

-Duser.timezone=<*TIMEZONE*>

where

TIMEZONE is your local Java time zone (e.g. Europe/Berlin for German time (GMT+1).)

Appendix

A Sample log4j configuration file

Starting with version 2.0.0 of this supportPac log4j version 1.x is no longer supported. Log4 version 2 needs a different configuration file.

The following is a sample log4j configuration file that can be used as a starting point. You can also find a sample as *brokerlog.xml* in the *samples* folder of this SupportPac.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration  monitorInterval="10" strict="false" packages="com.ibm.log4j.filter">
  <Appenders>
    <File name="FILE" fileName="C:/temp/Log4jTest.log" immediateFlush="true" append="true">
        <PatternLayout pattern="%d{yyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
      <ThresholdFilter level="WARN"/>
    </File>
  </Appenders>

  <Loggers>
    <Root>
        <AppenderRef ref="FILE"/>
    </Root>
  </Loggers>

</Configuration>
```