WebSphere Message Broker

Get Environment Variables Node

Version 3.0

3rd April, 2013

Note:
Before using this information and the product it supports, read the information in the "Notices" on page iv.

Fourth Edition (April 2011)

This edition applies to version 3.0 of WebSphere Business Integration Message Broker -- Get Environment Variables Node and to all subsequent releases and modifications until otherwise indicated in new editions.

Third Edition (September 2008)

This edition applies to version 2.0 of WebSphere Business Integration Message Broker -- Get Environment Variables Node and to all subsequent releases and modifications until otherwise indicated in new editions.

Second Edition (December 2006)

This edition applies to version 1.1 of WebSphere Business Integration Message Broker -- Get Environment Variables Node and to all subsequent releases and modifications until otherwise indicated in new editions.

First Edition (October 2005)

This edition applies to version 1.0 of WebSphere Business Integration Message Broker -- Get Environment Variables Node and to all subsequent releases and modifications until otherwise indicated in new editions.

Table of Contents

Notices

Trademarks and service marks

Acknowledgements

The author would like to acknowledge Arjan van Vught and his work on SupportPac IA06, which this SupportPac is based on.

Summary of Amendments

Date	Changes
27 September 2005	Initial release
24 September 2008	2.0 release
6 April 2011	3.0 release
3 April 2013	3.0 update for v8.0

Preface

This SupportPac supplies a WebSphere Business Integration Message Broker plug-in node that provides access to system environment variables, Java properties and message broker environment variables.

This SupportPac is based on SupportPac IA06 (GetEnvPlugin), but provides several enhancements and new features. Most notable are:

- Java Properties and Message Broker runtime variables are now exposed
- System, Java and Message Broker variables are created under the static Environment syntax tree
- The node implementation is Java (z/OS is not supported)
- The node provides tracing capability
- The plug-in is Eclipse based and compatible with Message Broker Toolkit v6.1, Message Broker Toolit v7.0 and Message Broker Toolkit v8.0.

Note, a separate Eclipse plug-in is available for v6.1 and for v7.0/v8.0.

This plug-in cannot be used as a direct replacement for the GetEnvPlugin node in migrated WMQI Version 2 message-flows. Because this plug-in utilizes the Environment syntax tree, not the Message syntax tree, environment variable correlation paths, in the migrated message-flows, are no longer valid. Replacing GetEnvPlugin nodes with this node will require modifications to the migrated message-flow.

Bibliography

- WebSphere Message Broker Version 8 Release 0
  Installation
  IBM Corporation.
- WebSphere Message Broker Version 8 Release 0
  User-defined Extensions
  IBM Corporation.
- WebSphere Message Broker Version 8 Release 0
  ESQL
  IBM Coporation.
- WebSphere Message Broker Version 7 Release 0
  Installation
  IBM Corporation.
- WebSphere Message Broker Version 7 Release 0
  User-defined Extensions
  IBM Corporation.
- WebSphere Message Broker Version 7 Release 0
  ESQL
  IBM Coporation.
- WebSphere Message Broker Version 6 Release 1
  Installation
  IBM Corporation.
- WebSphere Message Broker Version 6 Release 1
  User-defined Extensions
  IBM Corporation.
- WebSphere Message Broker Version 6 Release 1
  ESQL
  IBM Coporation.

Installing the plug-in node

You'll need to extract the contents of the IA9C.zip archive to a temporary location. The following sections will describe the installation of contents from the temporary location. Once your installation is complete, you may remove the contents of the temporary location.

SupportPac contents

The supplied zip file should be unzipped to a temporary directory. The following subdirectories and files will be created in an IA9C directory:

com.ibm.supportpacs.GetEnv – this directory contains the Eclipse plug-in for Message Broker Toolkit v6.1

com.ibm.supportpacs.GetEnv_3.0.0.jar – this JAR file contains the Eclipse plug-in for Message Broker Toolkit v7.0 and v8.0

\jpluginx – this directory contains the GetEnvNode implementation file, GetEnvNode.jar for WMB v6.1, v7.0 and v8.0

\sample – this directory contains the artifacts to install and run the sample message-flow

IA9C.pdf – the GetEnv plug-in user's manual

Prerequisites

This SupportPac provides a plug-in node to be used with WebSphere Message Broker Version v6.1, v7.0 or v8.0. For normal use, there are no other prerequisite products other than those required by IBM WebSphere Message Broker 6.1, 7.0 or 8.0 themselves.

Supported Platforms

This SupportPac is supported on the Windows XP, Windows Server, Windows 7, AIX, Sun Solaris, HP-UX and Linux platforms. The GetEnv node does not support z/OS.

Installing the plug-in node on the Broker system

Perform the following tasks on each runtime broker you want to install the GetEnv node:

1. Stop the broker on which you will install the GetEnv node.
2. Create a directory if you haven't already got one for this purpose. Add the directory to the LILPATH by using the **mqsichangebroker** command.

3. Put GetEnvNode.jar file in the directory, and make sure that the broker has access to it.
4. Start the broker to implement the change and to ensure that the existence of the new file is detected.
5. **CAUTION: Do not put the GetEnvNode.jar file in the WebSphere Message Broker installation directory, because it could be overwritten by the broker.**

Integrating the plug-in node into the Message Brokers Toolkit

This SupportPac provides an Eclipse plug-in for the following versions of the Message Broker Toolkit:

- IBM WebSphere Message Broker Toolkit version 6.1
- IBM WebSphere Message Broker Toolkit version 7.0
- IBM WebSphere Message Broker Toolkit version 8.0

All of the necessary files for integrating the plug-in with the Message Broker Toolkit 6.1 are contained under the following directory:

- com.ibm.supportpacs.GetEnv

For Message Broker Toolkit 6.1 copy the com.ibm.supportpacs.GetEnv directory to the following toolkit directory:

- <wmbtk_root>\WMBT610\plugins, where <wmbtk_root> is the root installation directory for your Message Broker Toolkit v6.1.

For Message Broker Toolkit 7.0 copy com.ibm.supportpacs.GetEnv_3.0.0.jar to the following toolkit directory:

- <wmbtk_root>\WMBT700\plugins, where <wmbtk_root> is the root installation directory for your Message Broker Toolkit v7.0.

For Message Broker Toolkit 8.0 copy com.ibm.supportpacs.GetEnv_3.0.0.jar to the following toolkit directory:

- <wmbtk_root>\WMBT800\plugins, where <wmbtk_root> is the root installation directory for your Message Broker Toolkit v8.0.
- NOTE: For WMB Toolkit v8.0.0.1 the installation path will be <wmbtk_root>\WMBT8001\plugins

You must stop and restart the Message Brokers Toolkit for the GetEnv plug-in to be available on the plug-in palette in the Message Flow Editor.

The GetEnv node is contained in the IBM SupportPacs plug-in category on the node palette.

Using the plug-in node

Description

The GetEnv plug-in node supports three Environment Variable groups:

- System – the operating system environment variables
- Java – the Java runtime property variables
- Message Broker – the Broker runtime environment variables

The Environment Variable groups are selectable, and any combination of them can be selected.

Each Environment Variable group has a corresponding sub-tree in the Environment syntax tree, as shown below:

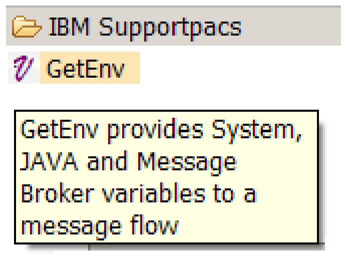
- System – Environment.ENV
- Java – Environment.ENV.JAVA
- Message Broker – Environment.ENV.MB

The System and Java variables are dynamic and are determined by the operating system environment and the Java runtime environment. The Message Broker variables are fixed.

The selected environment variable groups are initialized once, when the node is invoked on the first invocation of the message-flow. The node will not re-initialize modifications to the System and Java environment variables until the broker is re-started, or the message-flow is re-deployed.

Plug-in category

When installed, the GetEnv plug-in will appear in the IBM SupportPacs plug-in category in the Message Brokers Toolkit's Message-Flow editor.



GetEnv provides System, JAVA and Message Broker variables to a message flow

Plug-in node terminals

Terminal	Description
In	The input terminal that accepts a message assembly for processing by the node
Out	The output terminal that outputs a message assembly with a modified Environment syntax tree that now contains environment variable sub-trees
Failure	The output terminal that the message assembly is routed to if a failure occurs in the node

Plug-in node properties

The node's properties are displayed when you select Properties from the node's context menu.

Basic Group

Convert

This property specifies the conversion that will be applied to the environment variable names. The conversion is applied to all selected Environment Variable Groups. The default setting is No Conversion.

Selectable values are:

No Conversion

No conversion is performed.

Upper Case

All environment variable names are converted to upper case.

Lower Case

All Environment variable names are converted to lower case.

Use System Variables

This property specifies that System environment variables be created under the Environment.ENV parent element. The default setting is false.

Use JAVA Variables

This property specifies that the Java runtime properties be created under the Environment.ENV.JAVA parent element. The default setting is false.

Use Message Broker Variables

This specifies that Message Broker runtime variables be created under the Environment.ENV.MB parent element. The default setting is false.

Trace Group

Trace Node Activity

Specifies that node activity be traced, either to the broker's runtime console, or the file specified in the Trace Output To File property. If no file is specified in Trace Output To File, then output is to the broker's console. The broker must be configured to display a console. See the Configuring Broker Console Display section for instructions on this configuration.

The default setting is false.

This property is configurable in the broker archive file.

Trace Output To File

This property specifies the file to write the node's trace output. The Trace Node Activity property must be selected for trace output to be written to the named file. If the trace file exists, trace output will be appended. An exception is thrown if the trace file cannot be opened, or written to. Enter a single space (no quotes) to trace output to the Execution Group console (see "Execution Group Console Display" to locate the Execution Group's console.txt file).

The default setting is a single space character.

This property is configurable in the broker archive file.

The node retrieves the selected environment variable groups only once, when the node is activated for the first time in the message-flow. Modifications to the node's properties take effect on a message-flow redeploy. If you want the node to re-initialize with new or different environment variables, then the deployed message flow must be re-started.

The Environment.ENV sub-tree will contain all of the system environment variables for the message broker's user account. As an extension, the plug-in provides an environment variable named ElapsedTime, which contains the node's execution time in milliseconds. The Environment.ENV.ElapsedTime element is always created, even if no environment variable groups are selected.

The Environment.ENV.JAVA sub-tree will contain all of the Java properties that are defined for the message broker's Java runtime environment. Natively, the Java properties use periods as name separators – java.vm.specification.vendor – which are an inconvenience in ESQL code. The periods are replaced with an underscore before the syntax element is created – java_vm_specification_vendor.

The Environment.ENV.MB sub-tree will contain the following variables:

- NodeName – the name of the runtime node
- CoordinatedTransaction – “TRUE” or “FALSE”
- DataSourceUserId – the datasource user id that is used by the broker
- BrokerName – the name of the runtime broker
- ExecutionGroupName – the name of the runtime Execution Group
- QueueManagerName – the name of the runtime broker’s queue manager
- MessageFlowName – the name of the runtime message-flow
- ThreadName – name of the thread that invoked the GetEnv node

Plug-in best practices

A message-flow should contain only a single GetEnv node, on each distinct path of control. Because the environment variables are created under the static Environment syntax tree, they are available on any reachable path of control within the message-flow, once they have been created. So, placing a GetEnv node as the first node after the input node makes the environment variables available on any subsequent path of control within the message-flow.

An example would be to place a GetEnv node directly after an MQInput node. Now consider the Failure path of an MQInput node. In this case, the original GetEnv node will not be reached, as the failure occurred within the MQInput node. If the message-flow developer wishes to reference the MessageFlowName, ExecutionGroupName, and BrokerName while handling the failure, another GetEnv node will have to be placed on the MQInput node’s Failure path to obtain the message broker environment variables.

A main consideration is to avoid recreation of environment variable sub-trees within a single invocation of a message-flow. The overall cost of a GetEnv node is proportional to the number of elements that are created in the environment variable sub-trees. The environment variables are retrieved once, at node initialization, and cannot be changed until the broker is recycled. Multiple GetEnv nodes, on a single path of control within a message-flow, all create identical internal storage structures and environment variable sub-trees, and this will negatively impact message-flow performance. The node does not reuse a previously built environment variable sub-tree. If one is detected, it is detached and a new sub-tree is built. So, it is best to use one GetEnv node to obtain all required environment variable groups, rather than multiple GetEnv nodes to obtain the environment variable groups in stages.

Another consideration is to avoid using environment variable groups that are not referenced. The System environment variables can contain hundreds of variables that may have long values assigned to them. If none of the System environment variables are referenced, then the Use System Variables property should be unchecked. Likewise, the number of System environment variables could be reduced by removing any unused variables.

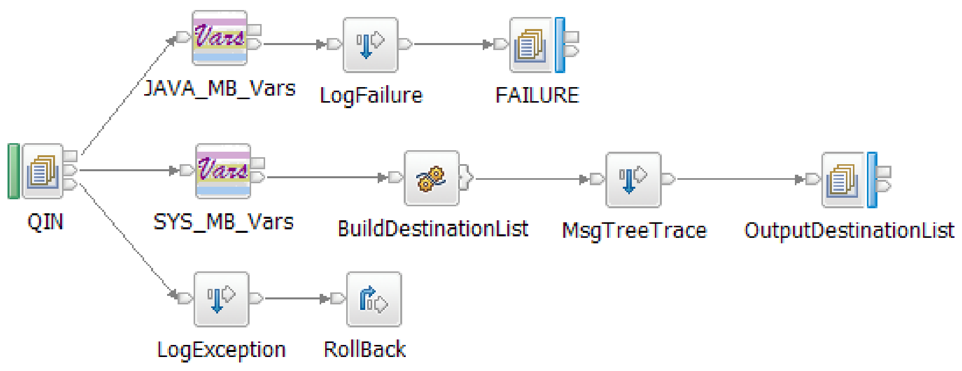
The ElapsedTime environment variable can be used to determine the performance of a GetEnv node. The ElapsedTime variable is always created under the Environment.ENV sub-tree, and contains the node's execution time in milliseconds.

The Trace Node Activity property should only be used for debugging or informational purposes, and should not be used in production situations.

Example using the GetEnv node

The IA9C_Sample message-flow demonstrates a possible use for the GetEnv plug-in. In this example the System, Java and Message Broker environment variable groups will be used. The example message-flow contains two GetEnv nodes – one on the main path of control and a second on the input node's failure path.

The example message-flow is illustrated below:



An XML input message is received by the QIN MQInput node.

The SYS_MB_Vars GetEnv node is used to create the System and Message Broker environment variable groups, under the Environment.ENV element. Note, that the environment variable names are converted to upper case, and no tracing is performed. This means that an environment variable name of any case will be converted to upper case.

BuildDestinationList is a Compute node that constructs a dynamic MQ DestinationList using two variables. The first variable is REGION, which defines a regional location such as EAST or WEST. The second variable is a routing list that contains a list of MQ queue destinations, delimited by ";" (example: Q1;Q2;Q3). The name of the routing list variable is the value of the REGION variable, concatenated with "RTLIST" (example: if REGION=EAST, then the routing list variable name is EASTRTLIST).

Note that the BuildDestinationList compute node's Compute Mode is set to "All", as the node is generating new Message and LocalEnvironment syntax trees.

The value of REGION can be defined in the input XML message, or in a system environment variable, as follows:

- If the input XML tag – InputRoot.XMLNSC.Msg.Region – exists, it is used to determine the name of the routing list environment variable

- If the input XML tag – InputRoot.XMLNSC.Msg.Region – doesn't exist, then the element – Environment.ENV.REGION – is used to determine the name of the routing list environment variable
- If neither InputRoot.XMLNSC.Msg.Region nor Environment.ENV.REGION exists, then a default routing list of DEFAULTQUEUE is used.

The routing list System environment variable should exist, and contain a valid list of MQ queue destinations. If the routing list System environment variable doesn't exist a default queue of DEFAULTQUEUE will be used. If the routing list System environment variable does exist but doesn't contain a list of valid queue destinations, then the OutputDestinationList MQOutput node will throw an exception.

MsgTreeTrace is a Trace node that logs the contents of the Message, LocalEnvironment and Environment syntax trees. You must provide a valid trace file for this node.

OutputDestList is an MQOutput node whose Destination Mode is set to Destination List.

On the catch path, the LogException Trace node will log the exception supplying the message-flow, execution group and broker names. These variables are made available by the SYS_MB_Vars node. Note that the environment variable element names are in upper case, because the GetEnv node Convert property is set to Upper Case. You must provide a valid trace file for this node.

RollBack is a Throw node that re-throws the exception, causing a rollback.

On the failure path, the Java_MB_Vars node is configured to use the Java and Message Broker environment variables, no conversion is applied to the variable names and tracing is turned off. Note that the failure path requires its own GetEnv node to access the Java and Message Broker variables. The LogFailure Trace node will log the failure supplying the message-flow, execution groups and broker names, and some of the Java runtime properties. Note that the environment variable element names are in native case, because the GetEnv node is set to no conversion. You must provide a valid trace file for this node.

Failure is an MQOutput node that is used to store the original input message on a failure queue.

Following is the ESQL code from the BuildDestinationList Compute node that accesses the System environment variables:

```
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
        SET OutputRoot = InputRoot;

        DECLARE queueName, rtList CHARACTER;
        DECLARE idx1,idx2, loopIdx Int 1;
        DECLARE sysVars REFERENCE TO Environment.ENV;
        DECLARE regionEnvVar REFERENCE TO Environment.ENV.REGION;
        DECLARE regionXmlTag REFERENCE TO InputRoot.XMLNSC.Msg.Region;
        CREATE FIELD OutputLocalEnvironment.Destination.MQ;
        DECLARE outRef REFERENCE TO OutputLocalEnvironment.Destination.MQ;

        -- First check for Region in the XML message, then
        -- in the System environment variable.
        -- Use DEFAULTQUEUE if routing list can’t be determined.
        IF (regionXmlTag IS NOT NULL) AND (regionXmlTag <> '') THEN
                SET rtList = sysVars.{UPPER(regionXmlTag) || 'RTLIST'};
        ELSEIF (regionEnvVar IS NOT NULL) AND (regionEnvVar <> '') THEN
                SET rtList = sysVars.{UPPER(regionEnvVar) || 'RTLIST'};
        ELSE
                SET rtList = 'DEFAULTQUEUE';
        END IF;

        -- In case "REGION"RTLIST environment variable doesn't exist
        IF (rtList IS NULL) THEN
                SET rtList = 'DEFAULTQUEUE';
        END IF;

        -- Build the DestinationList from the contents of the
        -- routing list environment variable.
        -- Queue names are delimited by a ';' (semi-colon), the last queue
        -- name is not delimited (example: "Q1;Q2;Q3")
        WHILE (idx2 <> 0) DO
                SET idx2 = POSITION(';' IN rtList FROM idx1);

                IF (idx2 > 0) THEN
                        SET queueName = SUBSTRING(rtList FROM idx1 FOR
                                                              (idx2 - idx1));
                ELSE
                        SET queueName = SUBSTRING(rtList FROM idx1);
                END IF;

                SET outRef.DestinationData[loopIdx].queueName = queueName;

                SET loopIdx = loopIdx + 1;
                SET idx1 = idx2 + 1;
        END WHILE;

        RETURN TRUE;
END;
```

Notice that InputRoot.XMLNSC.Msg.Region is validated first. If this tag doesn’t exist in the input XML message, then the System environment variable – Region – is validated. Because the GetEnv node is configured to convert variable names to upper case, the correlation path to the Region environment variable is – Environment.ENV.REGION. If the InputRoot.XMLNSC.Msg.Region is set to “east”, or if the System environment variable Region is set to “east”, then a routing list environment variable named “EASTRTLIST” will be used. The correlation path to the routing list variable would be – Environment.ENV.EASTRTLIST.

Any value of Region is support, as long as there is an environment variable named "RegionValue"RTLIST (all upper case), and it contains a valid list of MQ queue destinations. If the value of Region or the routing list variable can't be determined, then the DEFAULTQUEUE queue is used. If the routing list variable contains an invalid queue destination list, then the MQOutput node will thrown an exception.

Following is the pattern ESQL code from the LogFailure Trace node that accesses Message Broker and Java environment variables. Note that the environment variable element names are in native case, as the GetEnv node is configured to not convert them:

```
**********************************************************
 Date: ${CURRENT_DATE}
 Time: ${CURRENT_TIME}

 A failure has occurred in the MQInput node in the following messageflow:
  MessageFlow: ${Environment.ENV.MB.MessageFlowName}
  ExecutionGroup: ${Environment.ENV.MB.ExecutionGroupName}
  Broker: ${Environment.ENV.MB.BrokerName}
  QMgr: ${Environment.ENV.MB.QueueManagerName}

  OS: ${Environment.ENV.JAVA.os_name}
  OS Version: ${Environment.ENV.JAVA.os_version}
  JRE: ${Environment.ENV.JAVA.java_fullversion}

  The original message has been routed to the FAILURE queue

**********************************************************
```

Following is the pattern ESQL code from the LogException Trace node that accesses the Message Broker environment variables. Note that the environment variable element names are in upper case, as the GetEnv node is configured to convert them to upper case:

```
 ******************************************************************************
 Date: ${CURRENT_DATE}
 Time: ${CURRENT_TIME}

 An Exception has been caught by the Catch terminal of the MQInput node in the following messageflow:
  MessageFlow: ${Environment.ENV.MB.MESSAGEFLOWNAME}
  ExecutionGroup: ${Environment.ENV.MB.EXECUTIONGROUPNAME}
  Broker: ${Environment.ENV.MB.BROKERNAME}
  QMgr: ${Environment.ENV.MB.QUEUEMANAGERNAME}

  Following is a dump of the exception that was caught:

  ${ExceptionList}

******************************************************************************
```

Following is the ESQL code from the MsgTreeTrace Trace node:

```
****************************************************
 Root: ${Root}

~~~

 LocalEnvironment: ${LocalEnvironment}

~~~
```

```
Environment: ${Environment}
*****************************************************
```

Installing the IA9C Sample Message-Flow

Prior to installing the IA9C sample the GetEnv node implementation file should be installed on your broker, and the appropriate GetEnv Eclipse plug-in should be installed on your Message Broker Toolkit.

The IA9C.zip archive contains the IA9C sample message-flow collateral.

Unzip the contents of IA9C.zip to a temporary location. The IA9C\sample directory contains all of the resources to install and run the sample message-flow.

The sample message-flow can be executed by importing the supplied Eclipse project interchange into your Message Broker Toolkit and rebuild and deploy the sample Message Broker Archive (.bar) file. The sample Eclipse project interchange also contains the sample message-flow and ESQL source file for you to observe.

MQ Setup

The IA9C sample message-flow using the following MQ queues:

- QIN – the message-flow's input queue
- FAILURE – the message-flow's failure queue
- EAST1 – an Eastern region queue
- east2 – another Eastern region queue
- WEST1 – an Western region queue
- west2 – another Western region queue
- DEFAULTQUEUE – the default queue, when a routing list cannot be determined

The IA9C\sample\mq\queues.mqsc file contains the MQ script command to create these queues. Execute this script file on your broker's queue manager to create the required queues by issuing the following instruction on a command line:

```
runmqsc QUEUEMANAGERNAME < queues.mqsc
```

File System Setup

The IA9C sample message-flow contains three Trace nodes that write to trace output files. By default these files are contained in the -- C:\Temp\IA9C directory. This directory must exist if you want to deploy the IA9C_Sample.bar file unchanged.

If you want to change the log file directory, then you must edit the IA9C_Sample.bar configuration. On the Configure tab of each Trace node, modify the "File Path" property appropriately.

If you intend to import the message-flow source files, and then deploy, you can modify the "File Path" property on each of the Trace nodes in the IA9C_Sample message-flow.

Importing the IA9C_Sample Project Interchange

The sample message-flow can be run by importing the IA9C/Sample/ProjectInterchange/IA9C_Sample_ProjectInterchange.zip project interchange into your WMB Toolkit workspace and deploying the IA9C_Sample.bar file located under the IA9C_Sample_MFPrj message-flow project. You may need to modify the trace file paths in the Trace nodes, if the default directory --C:\Temp\IA9C is not suitable.

Start your toolkit, activate the Broker Application Development perspective and perform the following steps:

- File->Import
- On the Import dialog expand the Other group, highlight Project Interchange and click Next
- Use the Browse button to select the IA9C/Sample/ProjectInterchange/IA9C_Sample_ProjectInterchange.zip file, select the IA9C_Sample_MFPrj message-flow project and select Finish

You can now deploy the IA9C_Sample bar file to your broker and run the sample message-flow.

Executing the IA9C_Sample Message Flow

The IA9C_Sample message-flow makes use of four system environment variables:

- region – will be set to ‘east’
- eastrtlist – will be set to ‘EAST1;east2’
- westrtlist – will be set to ‘WEST1;west2’
- farwestrtlist – will be set to ‘NOTAQUEUE’

These environment variables must be set prior to starting your broker, for the sample message-flow to function as described. The method of setting these environment variables is platform dependent.

Setting the Environment Variables on Windows Platforms

On the Windows platforms the System Properties dialog is used to set the environment variables. The variables must be created as System variables (they are accessible to any system account), or as a user account variable for the user account that your broker is configured to run under. You cannot use the “set” command to set the value of the environment variable at the command-line, and then issue the mqsistart command to start your broker. These environment variables must be set using the System Properties dialog as described below:

- Windows Window XP
    - Select Start->Settings->Control Panel
    - From Control Panel, select System, to display the System Properties dialog

  - On the System Properties dialog, select the Advanced tab
  - On the Advanced tab, select Environment Variables
  - Create New System variables
  - Or, if this is the account that your broker executes under, then you can make the variables user variables
  - Create the four environment variables, as listed above. Do not include in quotes in the variable value.

If you want to change the setting of the environment variables, then you must restart your broker for the new variable values to be used by the sample message-flow.

Setting the Environment Variable on UNIX Platforms

- AIX, Solaris, Linux and HP-UX
  - The "export" command is used to set these environment variables on UNIX platforms
  - At the command-line issue the "export" command for each of the environment variables, then issue the mqsistart command to start your broker
  - Use the "export" command to create the four environment variables as listed above. For eastrtlist and westrtlist, the values must be enclosed in double quotes, as the ';' delimiter has special meaning. (Example: export westrtlist="WEST1;west2")

If you want to change the setting of the environment variables, then you must restart your broker for the new variable values to be used by the sample message-flow.

Sending Input Messages and Tracking Output Messages

You can use MQ Explorer to send the sample input messages and track the output messages. If you have another MQ application that you'd like to use, that is fine.

Using XML Message Content and Environment Variable for Routing Information

It is assumed that you have successfully installed the GetEnv node and sample, and have successfully deployed the sample message-flow. As well, the four sample environment variables are assumed to be correctly set, and that your broker user account has access to them.

In this configuration of the sample message flow, the XML input message will define the "region", and the message flow will determine which environment variable contains the region's routing list – "eastrtlist" or "westrtlist". The MQ DestinationList is then constructed from the region's routing list variable.

Send the contents of the IA9C\sample\messages\RegionWest.xml file to the QIN input queue, using MQ Explorer or an application of your choice. This message defines the Region as "west", and the message-flow will use the "westrtlist" variable to get a list of queue destination – in this case the queues are WEST1 and west2. You should see the input message has been routed to these two queues. In addition, you should see the message trace in the "C:\Temp\IA9C\MsgTreeTrace.log" file. Following is an example of the message trace output:

```
**************************************************
 Root: (
 (0x01000000):Properties = (
   (0x03000000):MessageSet      = ''
   (0x03000000):MessageType     = ''
   (0x03000000):MessageFormat   = ''
   (0x03000000):Encoding        = 546
   (0x03000000):CodedCharSetId  = 437
   (0x03000000):Transactional   = TRUE
   (0x03000000):Persistence     = FALSE
   (0x03000000):CreationTime    = GMTTIMESTAMP '2004-10-16 18:47:16.150'
   (0x03000000):ExpirationTime  = -1
   (0x03000000):Priority        = 0
   (0x03000000):ReplyIdentifier = X'000000000000000000000000000000000000000000000000'
   (0x03000000):ReplyProtocol   = 'MQ'
   (0x03000000):Topic           = NULL
 )
 (0x01000000):MQMD       = (
   (0x03000000):SourceQueue      = 'QIN'
   (0x03000000):Transactional    = TRUE
   (0x03000000):Encoding         = 546
   (0x03000000):CodedCharSetId   = 437
   (0x03000000):Format           = '        '
   (0x03000000):Version          = 2
   (0x03000000):Report           = 0
   (0x03000000):MsgType          = 8
   (0x03000000):Expiry           = -1
   (0x03000000):Feedback         = 0
   (0x03000000):Priority         = 0
   (0x03000000):Persistence      = 0
   (0x03000000):MsgId            = X'414d5120563220202020202020202020e11e744120005f03'
   (0x03000000):CorrelId         = X'000000000000000000000000000000000000000000000000'
   (0x03000000):BackoutCount     = 0
   (0x03000000):ReplyToQ         = '                                                '
   (0x03000000):ReplyToQMgr      = 'V2                                              '
   (0x03000000):UserIdentifier   = 'mattc       '
   (0x03000000):AccountingToken  = X'16010515000000668971706 8d4ca56ba66690aea03000000000000000000000b'
   (0x03000000):ApplIdentityData = '                                '
   (0x03000000):PutApplType      = 11
   (0x03000000):PutApplName      = 'C:\etc\rfhutil.exe          '
   (0x03000000):PutDate          = DATE '2004-10-16'
   (0x03000000):PutTime          = GMTTIME '18:47:16.150'
   (0x03000000):ApplOriginData   = '    '
   (0x03000000):GroupId          = X'000000000000000000000000000000000000000000000000'
   (0x03000000):MsgSeqNumber     = 1
   (0x03000000):Offset           = 0
   (0x03000000):MsgFlags         = 0
   (0x03000000):OriginalLength   = -1
 )
 (0x01000010):XML        = (
   (0x01000000):Msg = (
     (0x01000000):Region = (
       (0x02000000): = 'West' )
     (0x01000000):Content  = (
       (0x02000000): = message content for West region goes here' )
   )
 )
)
```

```
~~~

 LocalEnvironment: (
 (0x01000000):Destination = (
   (0x01000000):MQ = (
     (0x01000000):DestinationData = (
       (0x03000000):queueName = 'WEST1' )
     (0x01000000):DestinationData = (
       (0x03000000):queueName = 'west2' )
   )
  )
)

~~~

 Environment: (
 (0x01000000):ENV = (
   (0x03000000):ICU_DATA              = 'f:\wmqi\nnsy\share\icu\data'
   (0x03000000):PROCESSOR_ARCHITECTURE = 'x86'
   (0x03000000):PCOMM_ROOT            = 'C:\Program Files\Personal Communications'
   (0x03000000):DB2TEMPDIR            = 'E:\SQLLIB\'
   (0x03000000):MQ_JAVA_INSTALL_PATH   = 'F:\mqseries\Java'
   (0x03000000):SYSTEMROOT            = 'C:\WINNT'
   (0x03000000):MQ_LIB                = 'F:\mqseries\java\lib'
   (0x03000000):TMP                   = 'C:\WINNT\TEMP'
   (0x03000000):NNSY_ROOT             = 'f:\wmqi\nnsy'
   (0x03000000):OS                    = 'Windows_NT'
   (0x03000000):PROMPT                = '$P$G'
   (0x03000000):WINDIR                = 'C:\WINNT'
   (0x03000000):SYSTEMDRIVE           = 'C:'
   (0x03000000):COMSPEC               = 'C:\WINNT\system32\cmd.exe'
   (0x03000000):PROCESSOR_IDENTIFIER   = 'x86 Family 15 Model 2 Stepping 4, GenuineIntel'
   (0x03000000):COMMONPROGRAMFILES     = 'C:\Program Files\Common Files'
   (0x03000000):PROGRAMFILES          = 'C:\Program Files'
   (0x03000000):DICONFIGDIR           = 'f:\WDIServer32\Bin'
   (0x03000000):DB2INSTANCE           = 'DB2'
   (0x03000000):TEMP                  = 'C:\WINNT\TEMP'
   (0x03000000):NUMBER_OF_PROCESSORS   = '1'
   (0x03000000):REGION                = 'west'
   (0x03000000):MQ_JAVA_DATA_PATH      = 'F:\mqseries'
   (0x03000000):OS2LIBPATH            = 'C:\WINNT\system32\os2\dll;'
   (0x03000000):PROCESSOR_LEVEL       = '15'
   (0x03000000):DB2PATH               = 'E:\SQLLIB'
   (0x03000000):CROSSWORLDS           = 'f:\wbiadapters'
   (0x03000000):IBM_JAVACOREDIR       = 'f:/wmqi/errors/'
   (0x03000000):EASTRTLIST            = 'EAST1;east2'
   (0x03000000):WESTRTLIST            = 'WEST1;west2'
   (0x01000000):MB                    = (
     (0x03000000):EXECUTIONGROUPNAME     = 'default'
     (0x03000000):DATASOURCEUSERID       = 'mattc'
     (0x03000000):MESSAGEFLOWNAME        = 'ia9c_sample'
     (0x03000000):COORDINATEDTRANSACTION = 'FALSE'
     (0x03000000):QUEUEMANAGERNAME       = 'V2'
     (0x03000000):BROKERNAME            = 'BK'
     (0x03000000):NODENAME              = 'SYS_MB_Vars'
     (0x03000000):THREADNAME            = 'Thread-12'
   )
   (0x03000000):ELAPSEDTIME           = '20'
  )
)

*****************************************************
```

Notice that the XML message contains a “Region” tag with the content -- “West”. The message-flow uses this value to determine the name of the routing list variable, which in this case is WESTRTLIST.

In the LocalEnvironment tree we can see that the destination list was built from the "westrtlist" environment variable.

In the Environment tree we can see the REGION and WESTRTLIST variables. Recall that these names have been converted to upper case by the SYS_MB_Vars GetEnv node. (Note that the longer environment variables have been removed for readability.)

Using Only Environment Variables for Routing Information

In this configuration of the sample message flow, the XML input message does not contain a "Region" tag, and the message flow will determine if there is a "Region" environment. In this case the "Region" environment variable is set to "east", so the EASTRTLIST environment variable will be used. The MQ DestinationList is then constructed from the region's routing list variable.

Send the contents of the IA9C\sample\messages\NoRegion.xml file to the QIN input queue, using MQ Explorer or an application of your choice. The EASTRTLIST routing list will be used and you should see that the input message has been routed to the two eastern region queues – EAST1 and east2.

In addition, you should see the message trace in the "C:\Temp\IA9C\MsgTreeTrace.log" file. The contents of this message trace will be identical as the previous message; except the contents of the XML folder, which now contains a "NoRegion" tag.

Using a Non-existent Routing List Environment Variable

In this configuration of the sample message flow, the XML input message contains a "Region" tag for which there is no corresponding routing list environment variable. The value of the "Region" tag is "FarEast", for which there should be a corresponding FAREASTRTLIST environment variable, which contains a list of MQ queue destinations.

Because there is no FAREASTRTLIST environment variable, the default queue list of DEFAULTQUEUE is used.

Send the contents of the IA9C\sample\messages\RegionFarEast.xml file to the QIN input queue, using MQ Explorer or an application of your choice. The FAREASTRTLIST routing list can't be located so DEFAULTQUEUE is used, and you should see that the input message has been routed to the DEFAULTQUEUE queue.

In addition, you should see the message trace in the "C:\Temp\IA9C\MsgTreeTrace.log" file. The contents of this message trace will be identical as the previous messages; except the contents of the XML folder.

Using Environment Variables on the Catch and Failure Paths

In this configuration of the sample message flow an ill-formed routing list variable will be used to cause an exception to be thrown.

The sample input message defines the "Region" as "FarWest". The message-flow attempts to use the FARWESTRTLIST environment variable, but the queue destination contains a non-existent queue named NOTAQUEUE. The message-flow will correctly build the MQ Destination list, but the MQOutput node will throw an exception, as the named queue does not exist.

The Catch path of the MQInput node will be taken, where an exception message is written to – C:\Temp\IA9C\Exception.log. The exception message will contain Message Broker environment variables that were established via the SYS_MB_Vars GetEnv node, which is located on the main path of control.

The Catch path contains a Throw node which re-throws the original exception, causing the message to rollback. The rollback will cause the input message to be placed back on the QIN input queue. The input message will be retrieved by the QIN MQInput node again, but this time the message retry count will be exceeded (as long as the QIN queues Backout Threshold count is set to 0), causing an error within the MQInput node.

The Failure path of the MQInput node is taken when this error occurs. The LogFailure Trace node logs a failure message to a trace file, which contains Java and Message Broker environment variables, so this path of control requires a GetEnv node.

The JAVA_MB_Vars GetEnv node appears just before the LogFailure node, so the Java and Message Broker environment variables will be accessible in the trace message.

The original input message will then be routed to the FAILURE queue.

Send the contents of the IA9C\sample\messages\RegionFarWest.xml file to the QIN input queue, using MQ Explorer or an application of your choice. The FARWESTRTLIST routing list is located and the MQ DestinationList is correctly built, but the MQOuput node will throw an exception. The original input message will be routed to the FAILURE queue.

In addition, content will be written to the catch log file – C:\Temp\IA9C\Exception.log, and the failure log file – C:\Temp\IA9C\Failure.log, as well as the message tree log file – C:\Temp\IA9C\MsgTreeTrace.log. Both the exception and failure log messages reference environment variables, though the Environment syntax trees that they reference were constructed by two different GetEnv nodes.

Follow is the content that will be written to the exception log file:

```
*************************************************************************************
 Date: 2004-10-21
 Time: 03:26:34.405

 An Exception has been caught by the Catch terminal of the MQInput node in the following messageflow:
```

```
 MessageFlow: 'IA9C_Sample'
 ExecutionGroup: 'default'
 Broker: 'BK'
 QMgr: 'V2'

 Following is a dump of the exception that was caught:

 (
(0x01000000):RecoverableException = (
 (0x03000000):File             = 'F:\build\S500_P\src\DataFlowEngine\ImbMqOutputNode.cpp'
 (0x03000000):Line             = 751
 (0x03000000):Function         = 'ImbMqOutputNode::evaluate'
 (0x03000000):Type             = 'ComIbmMQOutputNode'
 (0x03000000):Name             = 'ia9c_sample#FCMComposite_1_2'
 (0x03000000):Label            = 'ia9c_sample.OutputDestinationList'
 (0x03000000):Text             = 'Caught exception and rethrowing'
 (0x03000000):Catalog          = 'BIPv500'
 (0x03000000):Severity         = 3
 (0x03000000):Number           = 2230
 (0x01000000):RecoverableException = (
  (0x03000000):File             = 'F:\build\S500_P\src\DataFlowEngine\ImbMqOutputNode.cpp'
  (0x03000000):Line             = 846
  (0x03000000):Function         = 'ImbMqOutputNode::handleListMessage'
  (0x03000000):Type             = 'ComIbmMQOutputNode'
  (0x03000000):Name             = 'ia9c_sample#FCMComposite_1_2'
  (0x03000000):Label            = 'ia9c_sample.OutputDestinationList'
  (0x03000000):Text             = 'Root exception for destination list'
  (0x03000000):Catalog          = 'BIPv500'
  (0x03000000):Severity         = 3
  (0x03000000):Number           = 2635
  (0x01000000):RecoverableException = (
   (0x03000000):File     = 'F:\build\S500_P\src\DataFlowEngine\ImbMqOutputNode.cpp'
   (0x03000000):Line     = 1874
   (0x03000000):Function = 'ImbMqOutputNode::putMessage'
   (0x03000000):Type     = 'ComIbmMQOutputNode'
   (0x03000000):Name     = 'ia9c_sample#FCMComposite_1_2'
   (0x03000000):Label    = 'ia9c_sample.OutputDestinationList'
   (0x03000000):Text     = 'Failed to open queue'
   (0x03000000):Catalog  = 'BIPv500'
   (0x03000000):Severity = 3
   (0x03000000):Number   = 2604
   (0x01000000):Insert   = (
    (0x03000000):Type = 5
    (0x03000000):Text = ''
   )
   (0x01000000):Insert   = (
    (0x03000000):Type = 5
    (0x03000000):Text = 'NOTAQUEUE'
   )
   (0x01000000):Insert   = (
    (0x03000000):Type = 2
    (0x03000000):Text = '2'
   )
   (0x01000000):Insert   = (
    (0x03000000):Type = 2
    (0x03000000):Text = '2085'
   )
  )
 )
 )
)

***********************************************************************************
```

From the exception log entry it is easy to determine that the NOTAQUEUE queue could not be opened. The Message Broker variables that are referenced in the log entry describe exactly where in the broker domain the error occurred.

Also the system's event log will document the exception with a series of BIP messages. Using the example above, the system's log file will contain the following series of BIP messages – BIP2230, BIP2635 and BIP2604. The full sequence of BIP messages in this series is dependant on the platform and event log configuration.

Following is the content of the failure log message that is written:

```
************************************************************
 Date: 2004-10-21
 Time: 03:26:35.457

 A failure has occurred in the MQInput node in the following messageflow:
   MessageFlow: 'IA9C_Sample'
   ExecutionGroup: 'default'
   Broker: 'BK'
   QMgr: 'V2'

   OS: 'Windows 2000'
   OS Version: '5.0'
   JRE: 'J2RE 1.3.1 IBM Windows 32 build cn131-20021102 (JIT disabled)'

   The original message has been routed to the FAILURE queue

************************************************************
```

The failure log message utilizes both the Message Broker and Java environment variables.

Trace Ouput Format

This section describes the format of the GetEnv node's trace output.

The node trace will document the initialization of the node and the creation of the Environment.ENV sub-tree.

The initialization trace will only occur once, on the first invocation of the GetEnv node in the message-flow. The initialization trace documents the creation of the internal objects that are used to cache the environment variables for the three variable groups. The selected Convert operation is applied to the individual variable names at this stage, and the converted names are used throughout node tracing.

The normal call sequence, which constructs the Environment.ENV sub-tree, occurs after the node has been initialized. The normal trace sequence documents the creation of the Environment.ENV sub-tree from the internal cache. The trace sequence is Environment.ENV, Environment.ENV.JAVA and Environment.ENV.MB, if all groups are selected.

The general format of the initialization trace output is shown below. For completeness, all environment variable groups are selected. For simplicity, only one variable for each group is shown (System=OS, Java=os_name and MB=BrokerName). Creation of the remaining variables follows the same format.

```
[2005-01-25 14:49:41:312 PST] >>ENTER GetEnvNode:  evaluate():
[2005-01-25 14:49:41:312 PST]     >>ENTER GetEnvNode:  initNode():
[2005-01-25 14:49:41:312 PST]       GetEnvNode:  initNode():  createing SYSTEM vars for OS<windows 2000>
[2005-01-25 14:49:41:373 PST]         GetEnvNode:  initNode():  created SYSTEM,  Key=<OS>,  Value=<Windows_NT>
[2005-01-25 14:49:41:383 PST]       GetEnvNode:  initNode():  creating JAVA vars
[2005-01-25 14:49:41:463 PST]         GetEnvNode:  initNode():  created JAVA,  Key=<os_name>,  Value=<Windows 2000>
[2005-01-25 14:49:41:483 PST]       GetEnvNode:  initNode():  creating MB vars
[2005-01-25 14:49:41:483 PST]         GetEnvNode:  initNode():  created MB,  Key=<BrokerName>,  Value=<BK>
[2005-01-25 14:49:41:483 PST]     <<EXIT  GetEnvNode:  initNode():
[2005-01-25 14:49:41:483 PST]     >>ENTER GetEnvNode:  buildEnvTrees():
[2005-01-25 14:49:41:483 PST]       GetEnvNode:  buildEnvTrees():  created element Environment.ENV
[2005-01-25 14:49:41:483 PST]         >>ENTER GetEnvNode:  buildEnvTree():  using parent element Environment.ENV
[2005-01-25 14:49:41:483 PST]           GetEnvNode:  buildEnvTree():  created element Name=<OS>, Value=<Windows_NT>
[2005-01-25 14:49:41:493 PST]         <<EXIT  GetEnvNode:  buildEnvTree():
[2005-01-25 14:49:41:493 PST]         >>ENTER GetEnvNode:  buildEnvTree():  using parent element Environment.ENV.JAVA
[2005-01-25 14:49:41:493 PST]           GetEnvNode:  buildEnvTree():  created element Environment.ENV.JAVA
[2005-01-25 14:49:41:503 PST]           GetEnvNode:  buildEnvTree():  created element Name=<os_name>, Value=<Windows
2000>

[2005-01-25 14:49:41:503 PST]         <<EXIT  GetEnvNode:  buildEnvTree():
[2005-01-25 14:49:41:503 PST]         >>ENTER GetEnvNode:  buildEnvTree():  using parent element Environment.ENV.MB
[2005-01-25 14:49:41:503 PST]           GetEnvNode:  buildEnvTree():  created element Environment.ENV.MB
[2005-01-25 14:49:41:503 PST]           GetEnvNode:  buildEnvTree():  created element Name=<BrokerName>, Value=<BK>
[2005-01-25 14:49:41:503 PST]         <<EXIT  GetEnvNode:  buildEnvTree():
[2005-01-25 14:49:41:503 PST]       GetEnvNode:  buildEnvTrees():  created element Environment.ENV.MB.ThreadName
[2005-01-25 14:49:41:503 PST]     <<EXIT  GetEnvNode:  buildEnvTrees():
[2005-01-25 14:49:41:503 PST]   GetEnvNode:  evaluate():  created element Environment.ENV.ElapsedTime
[2005-01-25 14:49:41:503 PST]   GetEnvNode:  evaluate():  calling propagate()
[2005-01-25 14:49:41:503 PST]   GetEnvNode:  evaluate():  return from propagate()
[2005-01-25 14:49:41:503 PST] <<EXIT  GetEnvNode:  evaluate():
```

Notice that initNode() is called immediately after entering evaluate(), as this is the first invocation of the GetEnv node. First the System variables are retrieved and cached (these calls are in blue), then the Java properties are retrieved and cached (these calls are in green), and then the Message Broker variables are retrieved and cached (these calls are in orange). This initialization call sequence will only occur on the first invocation of the GetEnv node. The names shown here have been converted using the node's convert property, and will be used for the remainder of the node's life cycle.

After the node has been initialized, the Environment.ENV sub-tree is constructed for the selected environment variable groups. The construction of the Environment.ENV sub-tree is driven by buildEnvTrees().

buildEnvTrees() first creates the Environment.ENV parent element (in magenta). An existing Environment.ENV element is not reused, it is deleted if present. Then it calls buildEnvTree() to create each of the environment variable group sub-trees – System=Environment.ENV, Java=Environment.ENV.JAVA and MB=Environment.ENV.MB..

buildEnvTree() displays which parent element it is working with, and the creation of each child element. These calls are in blue for the System variables, green for the Java variables and orange for the Message Broker variables.

There are two more calls highlighted in magenta. These calls are unique in that their values can't be cached, rather, they must be calculated on each invocation of evaluate(). The first call creates the Environment.MB.ThreadName element – the thread name can't be determined until evaluate() is called. The second call creates the Environment.ENV.ElapsedTime variable – the elapsed time must be calculated for each invocation of evaluate().

Special Trace Notes

The GetEnv node, by default, will write startup messages to the Execution Group's console.txt console file. These messages are not part of the GetEnv Trace feature, and are not controlled by GetEnv node properties. These messages are described below:

- getNodeName() – displays the node name returned by the getNodeName() method. This verifies that the node implementation file has been located and is being introspected.
- set property methods – each call to a set property method (setTraceFile() for example) will display the value passed to the method
- get property methods – each call to a get property method (getTraceFile() for example) will display the value being returned

Following is the general format for trace output after the GetEnv node has been itialized.

```
[2005-01-25 14:50:24:655 PST]  >>ENTER GetEnvNode:  evaluate():
[2005-01-25 14:50:24:655 PST]        >>ENTER GetEnvNode:  buildEnvTrees():
[2005-01-25 14:50:24:655 PST]           GetEnvNode:  buildEnvTrees():  created element Environment.ENV
[2005-01-25 14:50:24:655 PST]              >>ENTER GetEnvNode:  buildEnvTree():  using parent element Environment.ENV
[2005-01-25 14:50:24:735 PST]                 GetEnvNode:  buildEnvTree():  created element Name=<OS>, Value=<Windows_NT>
[2005-01-25 14:50:24:745 PST]              <<EXIT  GetEnvNode:  buildEnvTree():
[2005-01-25 14:50:24:745 PST]              >>ENTER GetEnvNode:  buildEnvTree():  using parent element Environment.ENV.JAVA
[2005-01-25 14:50:24:745 PST]                 GetEnvNode:  buildEnvTree():  created element Environment.ENV.JAVA
[2005-01-25 14:50:24:745 PST]                 GetEnvNode:  buildEnvTree():  created element Name=<os_name>, Value=<Windows
2000>
[2005-01-25 14:50:24:745 PST]              <<EXIT  GetEnvNode:  buildEnvTree():
[2005-01-25 14:50:24:745 PST]              >>ENTER GetEnvNode:  buildEnvTree():  using parent element Environment.ENV.MB
[2005-01-25 14:50:24:745 PST]                 GetEnvNode:  buildEnvTree():  created element Environment.ENV.MB
[2005-01-25 14:50:24:745 PST]                 GetEnvNode:  buildEnvTree():  created element Name=<BrokerName>, Value=<BK>
[2005-01-25 14:50:24:745 PST]              <<EXIT  GetEnvNode:  buildEnvTree():
[2005-01-25 14:50:24:745 PST]           GetEnvNode:  buildEnvTrees():  created element Environment.ENV.MB.ThreadName
[2005-01-25 14:50:24:745 PST]        <<EXIT  GetEnvNode:  buildEnvTrees():
[2005-01-25 14:50:24:745 PST]     GetEnvNode:  evaluate():  created element Environment.ENV.ElapsedTime
[2005-01-25 14:50:24:745 PST]     GetEnvNode:  evaluate():  calling propagate()
[2005-01-25 14:50:24:745 PST]     GetEnvNode:  evaluate():  return from propagate()
[2005-01-25 14:50:24:745 PST]  <<EXIT  GetEnvNode:  evaluate():
```

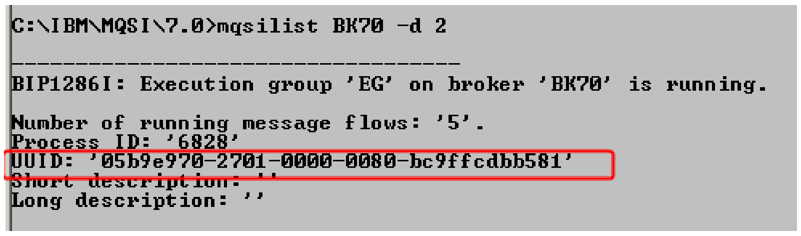
Notice that there is no call to initNode(). Processing begins by calling buildEnvTrees(), which creates the Environment.ENV elements and calls buildEnvTree() for each of the variable groups. The Environment.ENV.MB.ThreadName and Environment.ENV.ElapsedTime elements are created, and the message assembly is propagated.

Execution Group Console Display

The GetEnv node can be configured to send trace output to the Execution Group's console.txt file that it is deployed to. No configuration of the Broker or Execution Group is required.

You will need the Execution Group's UUID to locate its console.txt file. Issue the following mqsi command to determine the Execution Group ID:

- mqsilist BROKERNAME – d 2



```
C:\IBM\MQSI\7.0>mqsilist BK70 -d 2

-----------------------------------
BIP1286I: Execution group 'EG' on broker 'BK70' is running.

Number of running message flows: '5'.
Process ID: '6828'
UUID: '05b9e970-2701-0000-0080-bc9ffcdbb581'
Short description: ''
Long description: ''
```

The Execution Group's console.txt file will be located in the following location:

- {MQSI_WORKPATH}\components\{BKNAME}\{EG-UUID}\console.txt

Troubleshooting

The GetEnv plug-in node provides a number of features that aid in troubleshooting problems.

The node can generate a number of node specific exceptions, so the message broker system log should always be checked when an error, or unexpected results occur.

If the Flow Debugger is being used, then the node specific exceptions can be observed by examining the ExceptionList syntax tree.

The Trace Node Activity property can be used to troubleshoot usage problems with the node. When activated, this feature will trace, at node initialization time, the environment variables that were retrieved and the internal data structures that are built to store them. This feature will also display the sub-trees that are built when the node is accessed at runtime. Trace output can be sent to a file, or it can be displayed to the runtime broker's console.