MQSeries Integrator V2 - Postit Plug-In Version 2.4

11th September 2002

Mike Brady
Senior Consulting IT Specialist
IBM
Australia

mjbrady@au1.ibm.com

Take Note!

Before using this report be sure to read the general information under "Notices".

Sixth Edition, September 2002

This edition applies to Version 2.4 of *MQSeries Integrator V2 – Postit Plug-In* and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Notices

The following paragraph does not apply in any country where such provisions are inconsistent with local law.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program that does not infringe any of the intellectual property rights may be used instead of the IBM product.

Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, USA.

The information contained in this document has not be submitted to any formal IBM test and is distributed AS-IS. The use of the information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item has been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Trademarks and service marks

The following terms, used in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

- AIX
- IBM
- MQSeries
- MQSeries Integrator
- MQSI

The following terms are trademarks of other companies:

HP-UX	Hewlett Packard Corporation
Solaris	Sun Corporation
Windows NT	Microsoft Corporation

Summary of Amendments

Date	Changes
8th June 2001	Initial release
28th June 2001	Added support for sharing postits between Execution Groups in a broker.
3rd July 2001	Initial release as an MQSeries SupportPac
11th July 2001	Added support for Solaris platform
19th July 2001	Added support for Platform scope postits
29th August 2001	• Added support for HP-UX platform • Fixed bug when inserting data into XML messages
5th November 2001	• Added error handling for messages with missing MsgId and/or CorrelId • Added **AllMessages** match option to allow postits to match on any message
11th September 2002	• Fixed bug to allow PostitApply to work correctly when CopyDataTo is set to Root.BLOB • Fix alignment error in inflateSyntaxElement routine • Add preprocessor directives to allow source to build correctly against v2.1, v2.0.2 and v2.0.1 releases • Fix bug in Global postits (scope=Broker, scope=Platform) • Add new PostitDelete node • Fix bug that happens when Global and Local postits are mixed in the same Execution Group. • Added improved heap management logic and made garbage collection self-regulating • Shipped v2.0.1 and post v2.0.1 binaries

Preface

When implementing complex solutions with MQSeries Integrator V2, customers often find that they need to keep some form of context between message flows. A simple example of this need is found in the common request/reply message flows, where both the request message and the reply message need to be processed by the broker. Typically, the broker will manipulate the ReplyToQ field of the request message so that any reply message is redirected to a broker queue for further processing. The problem is then how to restore the ReplyToQ to its original setting so that the reply message can be sent to the originator of the request.

Various approaches to solving this problem exist, including storing the context in additional MQSeries queues or writing it to a database table.

The MQSI V2 Postit function provides an alternative mechanism for saving and restoring context information, without the use of either queues or database tables. It is quick, intuitive, simple to implement and is highly flexible in the ways it can be deployed. This function is implemented using a pair of cooperating nodes; the PostitCreate node and PostitApply node. The implementation allows context data to be shared in a variety of scopes, from Message Flows in the same Execution Group to Message Flows in any Execution Group in any Broker instance running on the same platform. Furthermore, it is multi-threaded so it will work equally well for deployments where Message Flows have been configured with multiple worker threads (Additional Instances > 0).

Chapter 1. Overview

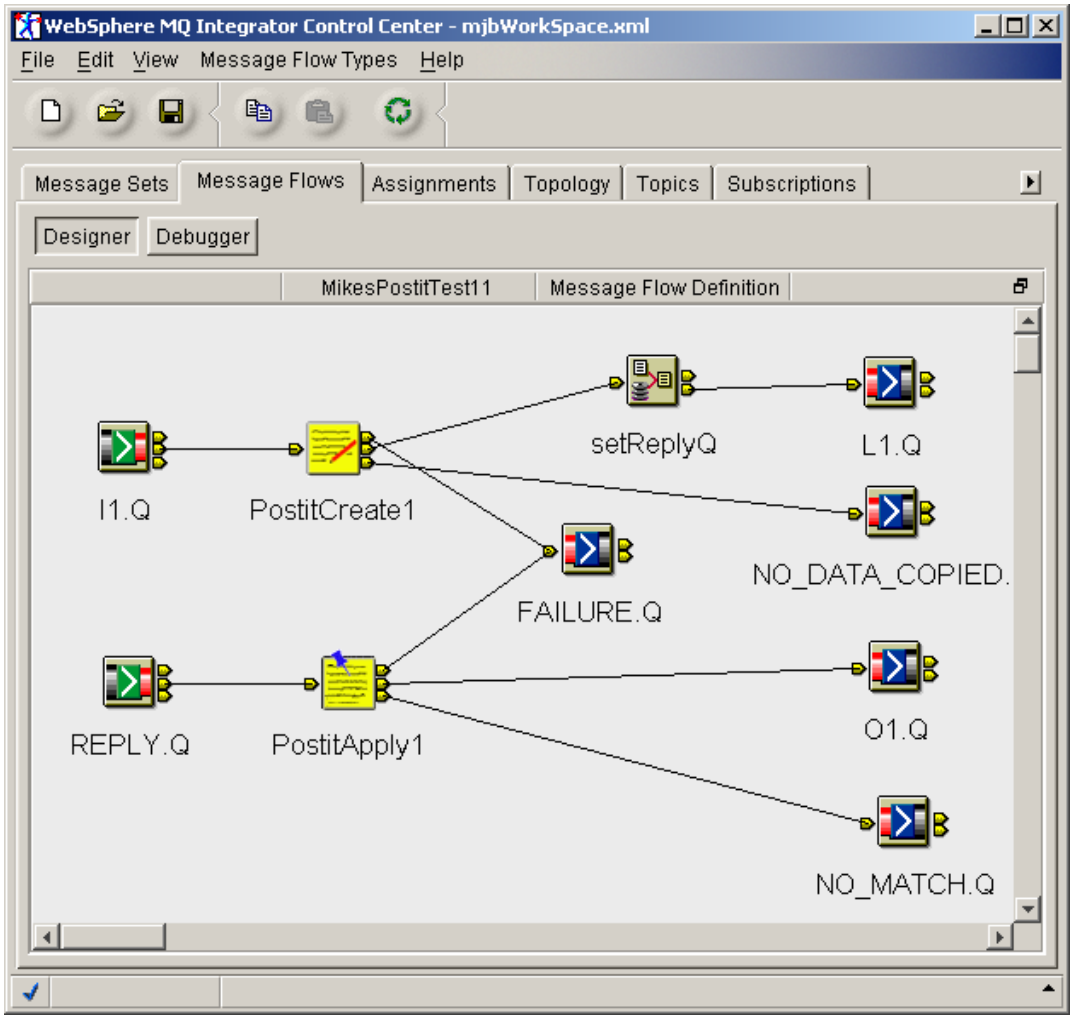
The Postit function is based on the concept of a virtual postit. (For anyone not familiar with the term, a 'postit' is a piece of paper, the original ones were yellow in colour, on which notes can be written. It is sticky-backed, enabling it to be attached to most surfaces.)

The PostitCreate node is used to create a virtual postit by copying data from a message into memory. The configuration of the PostitCreate node determines from which syntax tree element in the message the data will be copied from. The data can be any valid part of the syntax tree, from the contents of a single element, through to a complete tree structure. The resulting postit exists in memory and is available to be applied to a new message based on specified matching criteria.

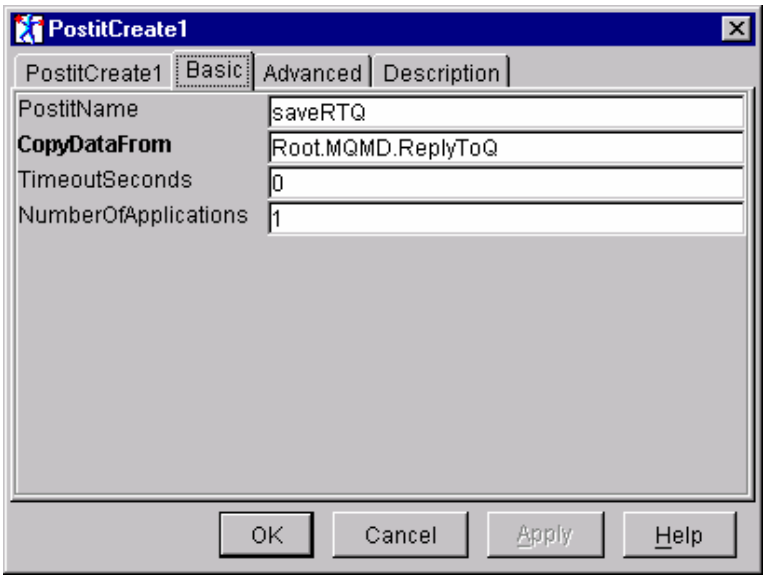
The PostitApply node is used to perform a search of the available postits and to apply (or 'stick') the data from any matching postit to the current message. The configuration of the PostitApply node determines the search criteria to be used to find matching postits. It also specifies the syntax tree element in the message to which the postit data should be copied.

The PostitDelete node can be used to remove existing postits.

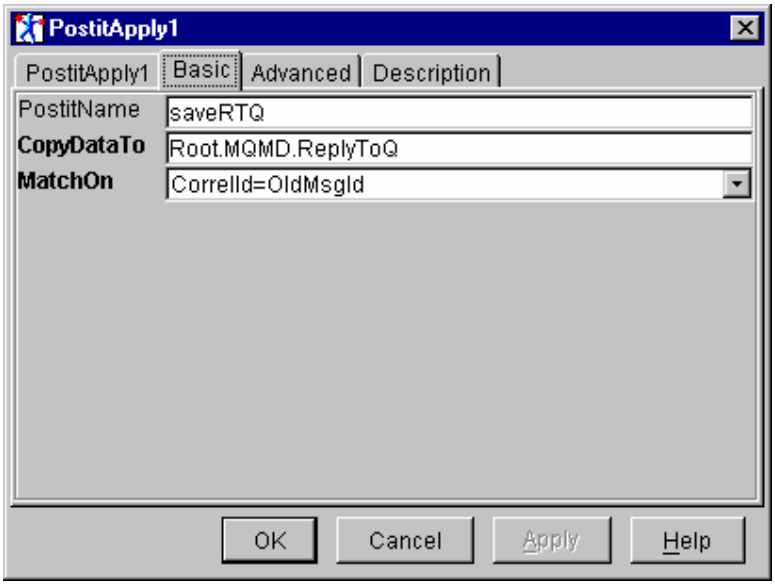
The following message flow shows a postit being created by passing the input message (I1.Q) through a PostitCreate node. The compute node(Compute1) resets the ReplyToQ to the value 'REPLY.Q' and the message is sent to a legacy system (L1.Q). The reply message (REPLY.Q) is then passed through a PostitApply node to restore the original setting of the ReplyToQ.



In this example, the use of the Postit function to save and restore the ReplyToQ involves manipulating a simple data value (the reply to queue name). The Properties window for the PostitCreate1 node is shown below. The node has been given the (optional) name 'saveRTQ' and has been directed to copy data from the syntax element 'Root.MQMD.ReplyToQ'. It has also been configured to so that it will be applied no more than once (NumberOfApplications=1) and it has an infinite lifetime (TimeoutSeconds=0).



The properties window for the corresponding PostitApply node (PostitApply1) is shown below. Should any matching postit exist, the data is to be copied from the postit into the syntax element described by the CopyDataTo property i.e. 'Root.MQMD.ReplyToQ'. The criteria for matching this message with available postits is determined by the setting of the MatchOn property and the optional PostitName property. In the example below, MatchOn is set to 'CorrelId=OldMsgId' which means that the CorrelId value of the current message must match the MsgId of the message which created the postit. The matching is further restricted by the PostitName property which is set to 'saveRTQ'. This reduces the set of possible matching postits to those created with a PostitCreate node with the **same** PostitName property value.



The PostitName property enables multiple postits to be created for a single message and to be subsequently applied to another message. For example, in the above message flow, a second pair of postit nodes could be introduced to save/restore the ReplyToQMgr value. Such a pair of nodes might have their PostitName property set to 'saveRTQM'.

The example shown involves manipulating a simple data type (the queue name), however, more complex data structures can be both saved and restored using the Postit function. For example, assume the input request message and the corresponding reply message contain the following XML.

Request Message

```
(0x1000010)XML       = (
   (0x1000000)request = (
     (0x1000000)requestTag1  = (
       (0x2000000) = 'request1'
     )
     (0x1000000)requestTag2  = (
       (0x2000000) = 'request2'
     )
     (0x1000000)requestTag3  = (
       (0x2000000) = 'request3'
     )
     (0x1000000)context = (
       (0x1000000)innerContext = (
         (0x1000000)contexTag1 = (
           (0x2000000) = 'context1'
         )
         (0x1000000)contexTag2 = (
           (0x2000000) = 'context2'
         )
       )
     )
   )
 )
```

Reply Message

```
(0x1000010)XML       = (
   (0x1000000)reply = (
     (0x1000000)replyTag1  = (
       (0x2000000) = 'reply1'
     )
     (0x1000000)replyTag2  = (
       (0x2000000) = 'reply2'
     )
     (0x1000000)replyTag3  = (
       (0x2000000) = 'reply3'
     )
   )
 )
```

The context data from the request message can be saved on a postit by configuring a PostitCreate node with CopyDataFrom set to 'Root.XML.request.context'. It could then be restored to the reply message by configuring a PostitApply node with CopyDataTo set to 'ROOT.XML.reply'. The resulting message with postit applied is shown in the following structure.

Reply Message with Postit applied

```
(0x1000010)XML       = (
   (0x1000000)reply = (
     (0x1000000)replyTag1  = (
       (0x2000000) = 'reply1'
     )
     (0x1000000)replyTag2  = (
       (0x2000000) = 'reply2'
     )
     (0x1000000)replyTag3  = (
       (0x2000000) = 'reply3'
     )
     (0x1000000)context = (
       (0x1000000)innerContext = (
         (0x1000000)contexTag1 = (
           (0x2000000) = 'context1'
         )
         (0x1000000)contexTag2 = (
           (0x2000000) = 'context2'
         )
       )
     )
   )
 )
```

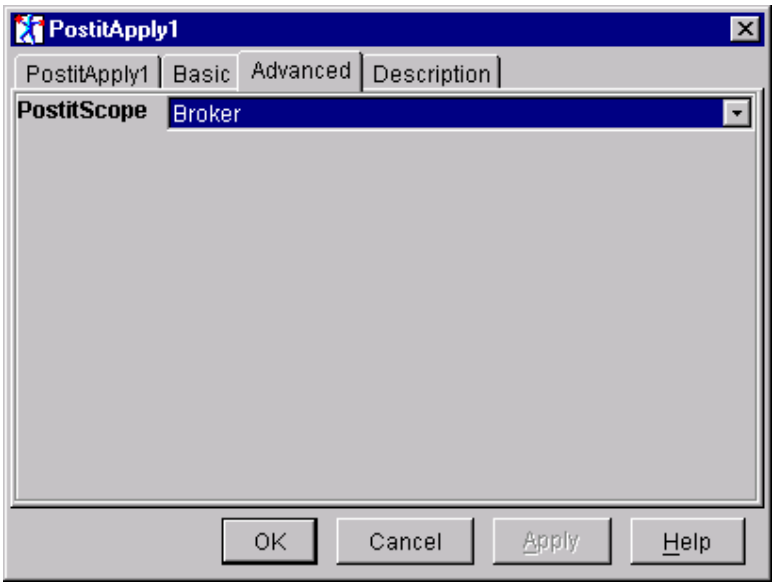
Note: The PostitApply node will attempt to insert data elements at the specified location, without regard to the integrity of the message. It is the responsibility of the Message Flow creator to ensure that the contents of the final message constitute a valid, well-formed message.

By default, a postit exists within the scope of the Execution Group. When processing a PostitApply node, the search is contained within this scope, i.e. only postits created by Message Flows within the same Execution Group are considered.

However, it is also possible to extend the scope of a postit so that it becomes visible to all Execution Groups within the scope of the Broker or all Execution Groups in all Brokers on the current platform. This option is available under the Advanced tag of the properties window. The PostitScope attribute allows either **ExecutionGroup**, **Broker** or **Platform** to be selected (the default is ExecutionGroup).

If the scope is set to Broker, then whenever a PostitApply node is executed, the search will include not only local postits, i.e. those created within the same Execution Group, but will also include any global postits created by other Execution Groups and having a scope value of Broker. If the scope is set to Platform, then the search will include postits created with Execution Group and Broker scope as well as those created with a scope value of Platform. In order for a postit to be categorised as global, it must have been created by a PostitCreate node with PostitScope=Broker or PostitScope=Platform.

The properties window below demonstrates one of the global settings of the PostitScope attribute.



By default, the size of the shared memory segment that will be allocated is 64 Kbytes. This is the minimum permitted size, however it can be increased should more storage be required. To modify the size of the shared segment, you must set the environment variable POSTIT_SHARED_MEMORY_KSIZE to the required number of Kbyte blocks.

You will probably need to run with your configuration and observe its behaviour in order to determine the optimal size of the shared memory segment, i.e. adjusting the shared memory size up or down until the system runs successfully with the minimum size segment. Some additional buffer should be included to allow for unexpected peaks in activity. Should the system run short on shared memory, a message will be written to the system log indicating this, and the Execution Group will be terminated.

Should you wish to calculate your shared memory requirements more methodically, as a rule of thumb, each global postit will require the following storage

- 200 bytes for control block structure
- storage for Execution Group name, calculated as length of Execution Group name + 1

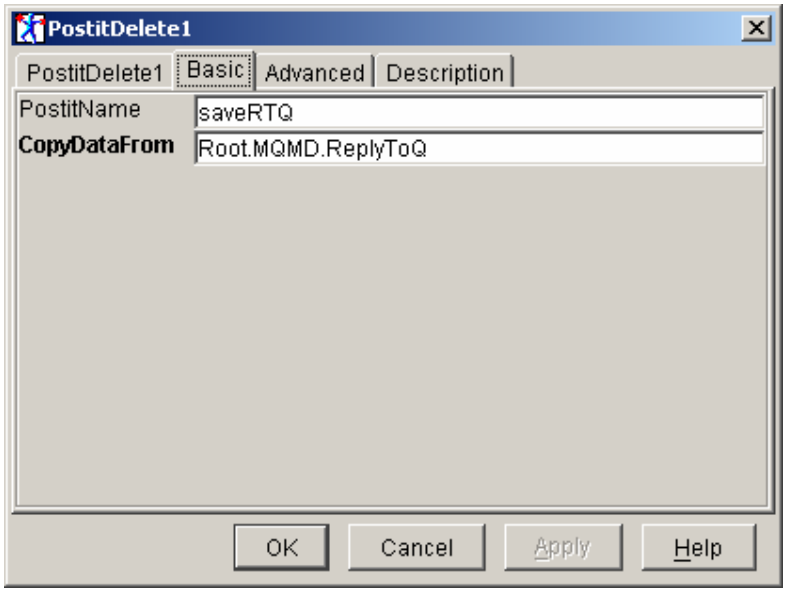
- storage for Message Flow name, calculated as (length of Message Flow + 1) x 2
- storage for the PostitName (if specified), calculated as (length of PostitName +1) x 2
- storage for saved data – this will be determined by the number and type of syntax element nodes that need to be saved. For example, saving the contents of the syntax element addressed by Root.MQMD.ReplyToQ will consist of

    1. control block data (approx. 70 bytes)
    2. the element name 'ReplyToQ'. This is stored in Unicode format and together with a field denoting its length. In this case it would require (9 x 2) + 4 bytes.
    3. data in element (24 bytes) together with its length field. In this case 24 + 4 bytes

The PostitDelete node provides a way of removing existing postits based on similar match criteria used by the PostitApply node. Whenever a PostitDelete node is executed, a search is made for postits which have the following matching attributes.

- PostitName
- CopyDataFrom
- PostitScope

To qualify for deletion, matching postits must also have been created by a message flow within the same execution group.



Chapter 2. Installation

SupportPac contents

The supplied zip file should be unzipped in a temporary directory. The following files and sub-directories will be created.

/source

- Makefile.NT
- Makefile.AIX
- postit.c
- plugin.c
- apply.c
- create.c
- delete.c
- alarm.c
- retcodes.c
- utilities.c
- postit.h
- postit_retcodes.h
- postit_constants.h
- plugin.h
- unicode.h
- threads.h
- rc.h
- trace_publ.h
- trace_defs.h
- trace_func.h
- trace_data.h
- postitRequest.c
- postitReply.c
- samples.mak.NT
- samples.mak.AIX
- samples.mak.Solaris

/NT/help

- MessageProcessingNodeType_PostitCreate.htm
- MessageProcessingNodeType_PostitApply.htm
- MessageProcessingNodeType_PostitDelete.htm

/NT/bin

- postit.lil
- v201/postit.lil
- postitRequest.exe
- postitReply.exe

/NT/messages

- MQSIV2_postit.msg
- MQSIV2_postit_msg.h
- MQSIV2_postit.dll
- Makefile

/NT/objects

- traceinit.obj
- trace.obj
- error.obj
- unicode.obj
- threads.obj
- heap.obj
- shmem.obj

/NT/config

- PostitCreate
- PostitCreate.wdp
- PostitApply
- PostitApply.wdp
- PostitDelete
- PostitDelete.wdp

/NT/images

- PostitCreate.gif
- PostitCreate30.gif
- PostitCreate42.gif
- PostitCreate58.gif
- PostitCreate84.gif
- PostitApply.gif
- PostitApply30.gif
- PostitApply42.gif
- PostitApply58.gif
- PostitApply84.gif
- PostitDelete.gif
- PostitDelete30.gif

PostitDelete42.gif
PostitDelete58.gif
PostitDelete84.gif

/AIX

/AIX/bin

postit.lil
v201/postit.lil
postitRequest
postitReply

/AIX/messages

MQSIV2_postit.cat
MQSIV2_postit.msg
MQSIV2_postit_msg.h

/AIX/objects

traceinit.o
trace.o
error.o
threads.o
unicode.o
heap.o
shmem.o

/Solaris

/ Solaris /bin

postit.lil
v201/postit.lil
postitRequest
postitReply

/ Solaris /messages

MQSIV2_postit.cat
MQSIV2_postit.msg
MQSIV2_postit_msg.h

/ Solaris /objects

traceinit.o
trace.o
error.o
threads.o
unicode.o

heap.o

shmem.o

/HPUX

/ HPUX /bin

postit.lil

v201/postit.lil

postitRequest

postitReply

/ HPUX /messages

MQSIV2_postit.cat

MQSIV2_postit.msg

MQSIV2_postit_msg.h

/ HPUX /objects

traceinit.o

trace.o

error.o

threads.o

unicode.o

heap.o

shmem.o

license2.txt

ia0h.pdf

Prerequisites

This SupportPac provides a Plug-in node to be used with the IBM MQSeries Integrator for Windows (NT or 2000) - V2.0.1, IBM MQSeries Integrator for AIX - V2.0.1, IBM MQSeries Integrator for HP-UX – V2.0.2 and IBM MQSeries Integrator for Solaris - V2.0.1 and above. For normal use, there are no other pre-requisite products other than those required by MQSeries Integrator V2.1 itself.

Supported platforms

This SupportPac has been developed for and tested on

- Microsoft Windows NT environment
- IBM AIX environment
- Sun Solaris environment
- HP-UX 11 environment

Installing the plug-in nodes on NT

1. Unzip the packaged files into a temporary directory.

2. Copy the message catalogue file **NT\messages\MQSIV2_postit.dll** to a directory of your choice, e.g. <MQSI root>\messages.

3. Add an entry for the message catalogue to the registry. Use regedit to add an entry to the registry under…

```
HKEY_LOCAL_MACHINE
    SYSTEM
      CurrentControlSet
        Services
          EventLog
             Application
```

   Create a new entry with the following details

```
MQSIV2_postit
  (default)             (value not set)
  EventMessageFile   <fully qualified name of MQSIV2_postit.dll>
  TypesSupported        0x00000007 (7)
```

4. Copy the file **NT\bin\postit.lil** to the MQSeries Integrator bin directory, e.g. <MQSI_root>\bin.

5. Set the POSTIT_SHARED_MEMORY_KSIZE environment variable if required.

6. Restart the broker and check for plug-in initialisation messages in Event log

Installing the plug-in nodes on AIX

1. Unzip the packaged files into a temporary directory.

2. Copy the message catalogue file **AIX\bin\MQSIV2_postit.cat** to a directory specified by the NLSPATH setting, e.g. <MQSI_root>/messages.

3. Copy the file **AIX\bin\postit.lil** to the MQSeries Integrator lil directory (<MQSI_root>/lil).

4. Set the POSTIT_SHARED_MEMORY_KSIZE environment variable if required.

5. Restart the broker and check for plug-in initialisation messages in the syslog.

Installing the plug-in nodes on Solaris

1. Unzip the packaged files into a temporary directory.

2. Copy the message catalogue file **Solaris\messages\ MQSIV2_postit.cat** to a directory specified by the NLSPATH setting, e.g. /usr/lib/locale/<*locale*>/LC_MESSAGES where <*locale*> is the locale under which the machine is running or 'C' if none is set.

3. Copy the file **Solaris\bin\postit.lil** to the MQSeries Integrator lil directory (<MQSI_root>/lil).

4. Set the POSTIT_SHARED_MEMORY_KSIZE environment variable if required.

5. Restart the broker and check for plug-in initialisation messages in the syslog.

Installing the plug-in nodes on HP-UX

1. Unzip the packaged files into a temporary directory.

2. Copy the message catalogue file **HPUX\messages\ MQSIV2_postit.cat** to a directory specified by the NLSPATH setting, e.g. /usr/lib/locale/<*locale*>/LC_MESSAGES where <*locale*> is the locale under which the machine is running or 'C' if none is set.

3. Copy the file **HPUX\bin\postit.lil** to the MQSeries Integrator lil directory (<MQSI_root>/lil).

4. Set the POSTIT_SHARED_MEMORY_KSIZE environment variable if required.

5. Restart the broker and check for plug-in initialisation messages in the syslog.

Integrating the plug-in node into the Windows Control Center

1. Unzip the packaged files into a temporary directory.

2. Change to the **NT\images** directory and copy its contents to <MQSI_root>\Tool\images

3. Change to the **NT\config** directory and copy its contents to <MQSI_root>\tool\repository\private\<machine name>\<Queue Manager name>\MessageProcessingNodeType

4. Change to the **NT\help** directory and copy its contents to <MQSI_root>\tool\help\com\isv

   Note: create this directory if it does not already exist.

5. Start the MQSeries Integrator Control Center and display the MessageFlows panel. Right click on IBM Primitives and select **Add to Workspace**, then **Message Flow**. Select the PostitCreate, PostitApply and PostitDelete nodes from the displayed list and add them to the palette.

6. Check in all new node types

Installation verification

Create a message flow that is similar to the one shown in the Overview section.

To simulate the backend legacy application, run the postitReply program. By default, this will process a message arriving on the legacy queues L1.Q. You can specify your own queue manager and input queue by invoking the application in the following way

postitReply <queueManagerName> <queueName>

Add messages to the input queue I1.Q by running the postitRequest program. This program takes a single parameter that specifies the number of input messages to be generated. Request messages are written to the I1.Q and replies are read from the O1.Q.

The source for both the postitRequest and postitReply programs is supplied along with an appropriate makefile for the target Operating System, e.g. samples.mak.NT to enable you to easily rebuild them.

Chapter 3. Nodes Reference

PostitCreate Node

A PostitCreate node copies the specified syntax element(s) and associated data into a virtual postit. The lifetime of the postit is determined by a timeout value and/or the maximum number of times it can be applied to a message.

It works in conjunction with a **PostitApply** node, storing relevant information from the message to enable subsequent matching to be performed by a PostitApply node.

Each message processed by a PostitCreate node will cause the PostitCreate node to perform the following actions:

- Create a new postit instance and save the Msgid and CorrelId
- Locate the syntax element specified by CopyDataFrom property. If not found, propagate the message to the noDataCopied terminal.
- Serialise the data at the located syntax element and store in postit in process storage if local postit or in shared memory storage if global postit
- Add the postit to required indexes
- If a TimeoutSeconds specified a non-zero value, add the postit to the alarm chain

PostitCreate node terminals	
Terminal	**Description**
in	The input terminal that accepts a message for processing by the node
out	The output terminal to which the message is normally routed
failure	The output terminal to which the message is routed if there is an error during the node processing
noDataCopied	The output terminal to which the message is routed if the specified syntax element cannot be found in the message.

PostitCreate node properties

These properties are displayed when you right click a PostitCreate node entry in the Message Flow Types pane, and click **Properties**. The values displayed are the default properties for this instance of the node. They cannot be edited when displayed from the Message Flow Types pane.

PostitName

This optional value makes it possible to associate a PostitCreate node with a PostitApply node (see the PostitApply node description for further details).

CopyDataFrom

Specifies the name of the syntax element from which the stored data will be copied. Names must begin with '**Root.**' otherwise an exception is raised at deployment time.

TimeoutSeconds

Specifies the lifetime (in seconds) of the postit. A value of 0 indicates an INFINITE lifetime. If a non-zero value is specified, an alarm is set to the timeout value and, if the postit still exists when the timer expires, it is automatically deleted.

NumberOfApplications

Specifies the maximum number of times the postit may be applied before being deleted. The default value is 1, which means that the postit will be deleted after its first application.

PostitScope

Enumerated value indicating the scope of the postit. The possible values are

- ExecutionGroup – the postit will visible to other nodes within the same Execution Group (this is the default value). In this case, postits will be stored in process memory.
- Broker – the postit will be visible to nodes in any Execution Group running within the Broker. In this case, postits will be stored in a shared memory segment.
- Platform – the postit will be visible to nodes in any Execution Group running within any Broker on the current platform. In this case, postits will be stored in a shared memory segment.

Configuring the PostitCreate node

For a description of the properties of the PostitCreate node and their possible values, see“PostitCreate node properties” above.

To configure an PostitCreate node:

1. In the Message Flow Definition pane, right click the symbol of the PostitCreate node you want to configure and click **Properties**, then click **Basic**. The **PostitCreate node** dialog is displayed.
2. In the **PostitCreate node** dialog, type values for those properties that you want to set.
3. If you want to provide a description of this instance of the PostitCreate node (which is recommended if you want other Control Center users to be able to make use of it), click the **Description** tab of the **PostitCreate node** dialog. Type a short description, or a long description, or both.
4. Click **OK** to finish configuring this PostitCreate node.

PostitApply Node

An PostitApply node is the corresponding node to the PostitCreate node.

It searches the available collection of postits, looking for any matching instances. The data from matching postits is then applied to the message.

Every message that passes into the PostitApply node will cause it to perform the following:

- Extract the Msgid and CorrelId
- Use the value specified by the MatchOn property to search the available postits looking for a match – only the local indexes are searched if the PostitApply node is local, otherwise the appropriate global indexes are also searched. For example if scope is Broker then the Broker index is searched and if scope is Platform, both the Broker and Platform indexes are searched.
- If PostitName was specified on the PostitApply node, further restrict the list of matching postits to those created by a PostitCreate node with a matching name
- If at least matching postit was found, copy the message and locate (or create if necessary) the syntax element specified by the CopyDataTo property
- If no matching postit was found, propagate message to the noMatch terminal

- Expand the serialised data from the postit and copy it into the specified syntax element
- If the postit specified NumberOfApplications, increment the number of times postit has been applied. If value equals NumberOfApplications, delete postit.
- Propagate message to the out terminal

If the node detects an error, the message will be routed to the **failure** terminal.

PostitApply node terminals	
Terminal	**Description**
in	The input terminal that accepts a message for processing by the node
out	The output terminal to which the message and applied postit data is routed
noMatch	The output terminal to which the message is routed if no matching postit is found
failure	The output terminal to which the message is routed if there is an error during the node processing

PostitApply node properties

These properties are displayed when you right click an PostitApply node entry in the Message Flow Types pane, and click **Properties**. The values displayed are the default properties for this instance of the node. They cannot be edited when displayed from the Message Flow Types pane.

PostitName

This optional value makes it possible to associate this PostitApply node with postits created by a PostitCreate node with the same name. This further restricts the matching criteria specified by the MatchOn property.

CopyDataTo

Specifies the name of the syntax element to which the stored data will be copied to. Names must begin with '**Root.**' otherwise an exception is raised at deployment time. If any of the elements in the path do not exist, they are created as a NAME-VALUE syntax element types.

MatchOn

Enumerated value indicating how the message attributes should be used to find matching postits. The possible values are

- CorrelId=OldMsgId – the CorrelId value of the current message must match the MsgId value of the original message that created the postit (this is the default value)
- MsgId=OldMsgId – the MsgId value of the current message must match the MsgId value of the original message that created the postit
- CorrelId=OldCorrelId – the CorrelId value of the current message must match the CorrelId value of the original message that created the postit
- MsgId =OldCorrelId – the MsgId value of the current message must match the CorrelId value of the original message that created the postit
- MsgId+CorrelId=OldMsgId+OldCorrelId – the combined MsgId and CorrelId values of the current message must match the combined MsgId and CorrelId values of the original message that created the postit

- AllMessages – the postit will be applicable to all messages processed by the PostitReply node, irrespective of the MsgId and CorrelId values. However, if specified, the value of the PostitName option will still be used to restrict the set of matches.

PostitScope

Enumerated value indicating the scope for matching against available postits. The possible values are

- ExecutionGroup – only the local indexes, i.e. those indexes that refer to postits created with PostitScope=ExecutionGroup will be searched.
- Broker – the local indexes, i.e. those indexes that refer to postits created with PostitScope=ExecutionGroup, and the global indexes, i.e. those indexes that refer to postits created with PostitScope=Broker will be searched.
- Platform – the local indexes, i.e. those indexes that refer to postits created with PostitScope=ExecutionGroup, and the global indexes, i.e. those indexes that refer to postits created with PostitScope=Broker or PostitScope=Platform will be searched.

Configuring the PostitApply node

For a description of the properties of the PostitApply node and their possible values, see"PostitApply node properties" above.

To configure an PostitApply node:

1. In the Message Flow Definition pane, right click the symbol of the PostitApply node you want to configure and click **Properties**.The **PostitApply node** dialog is displayed.
2. In the **PostitApply node** dialog, type values for those properties that you want to set.
3. If you want to provide a description of this instance of the PostitApply node (which is recommended if you want other Control Center users to be able to make use of it), click the **Description** tab of the **PostitApply node** dialog. Type a short description, or a long description, or both.
4. Click **OK** to finish configuring this PostitApply node.

Tracing the postit plug-in nodes

To trace execution of the plug-in nodes, set the POSTIT_PLUGIN_TRACE environment variable (system variable on NT) and reboot machine **before** restarting broker. Settings for trace are as follows.

POSTIT_PLUGIN_TRACE **=**

-f *traceOutputFileName* - name of file to write trace to. See below for further options.

-t - include time stamp on trace entries.

-i - include process and thread id on entries

-c - commit (flush) entries to file after every write

-l **-** trace level to output (see trace values below)

-a - append trace to existing trace file

Valid trace level settings are

- TRACE_NONE
- TRACE_ENTRY_EXIT
- TRACE_ERROR
- TRACE_WARNING
- TRACE_INFO
- TRACE_SYSTEM_ALL
- TRACE_ALL

The following setting results in comprehensive tracing and will be sufficient in most cases.

POSTIT_PLUGIN_TRACE=-f <name of output file> -i –c -l TRACE_SYSTEM_ALL

Substitution patterns may be used to generate more specific trace file names.

%P insert process id in file name (**%p** may also be used)

%T append thread id to file name (**%t** may also be used).

For example, the following setting

POSTIT_PLUGIN_TRACE=-f d:\\mqsilog\\trace%p.txt -i –c -l TRACE_SYSTEM_ALL

generates a trace file called d:\mqsilog\trace**nnnn**.txt where **nnnn** is the process id of the execution group process. This is useful for tracing postit plugins running in multiple execution groups.

End of Document