MQSeries Integrator
Version 1 to Version 2 Migration

This document addresses the migration of MQSeries Integrator installations from Version 1 to Version 2. It also addresses the issues associated with using Version 1 function (NEON Rules and Formats) within the Version 2 (message flow) environment. It describes what can be accomplished and, where necessary, the workarounds necessary to achieve this.

Supported upgrade and migration options

Migration to MQSeries Integrator is supported from the following products:

- MQSeries Integrator Version 1.0
- MQSeries Integrator Version 1.1

It is strongly recommended that you uninstall your existing MQ Integrator or MQSeries Integrator product prior to installing MQSeries Integrator Version 2. Before uninstalling you should make a backup copy of your Rules and Formatter definitions either by backing up the database tables directly or using the export facilities (NNRie and NNFie) You should also make backup copies of your configuration files (which, if you have kept the original defaults. will comprise: **sqlsvses.cfg, MQSIruleng.mpf, MQSIgetdata.mpf and MQSIputdata.mpf**) as these can be reused with MQSeries Integrator Version 2.

The MQSeries Integrator Version 2 installation program will detect whether Version 1.0 or Version 1.1 is already installed and if found will remove the Version 1.x path from the system path definitions so that the Version 1.x system is no longer accessible. The installation program will also copy the Version 1.x configuration files to the Version 2 install directory.

Although MQSeries Integrator Version 2 currently supports only IBM DB2 and Microsoft SQL Server databases, the NEON rules and formats continue to be supported in the same set of databases as with Version 1.1. If your V1.1 database is not one of those supported by MQSeries Integrator Version 2 then you may wish to consider migrating the database tables to one of the supported databases for consistency with the rest of the product. However, this is not necessary,

It is also important to note that the NEON rules and formats are not defined centrally and distributed automatically as is the case with the other MQSeries Integrator Version 2 configuration data. The NEON Rules and NEON Formatter tools must be run run directly against the operational data sets (as they are in Version 1.x).

It is possible to upgrade MQIntegrator (NEON product) to MQSeries Integrator Version 2 but the installation program will not detect the presence of MQIntegrator if it is already installed on your system. You must make sure therefore that you uninstall it before installing Version 2. The migration steps are identical with those required for migrating from MQSeries Version 1.0 to MQSeries Version 2.

Migrating your existing rules and formats

There are no specific actions required to migrate your existing rules and formats to MQSeries Integrator Version 2 from Version 1.1. If migrating from Version 1.0 you must first export your rules and format definitions using the NNRie and NNFie utilities and then import these definitions into the new database after installing Version 2. (For further information on using the NNRie and NNFie import/export utilities, refer to “Using the NEON Rules and Formatter in MQSeries Integrator Version 2”.

After installation you must make any necessary modifications to your MQSIruleng.mpf file (which will have been copied to the bin subdirectory of the directory in which you have installed the product. The installation program will also have created the MQSI_PARAMETERS_FILE environment variable to indicate the location of this file. MQSeries Integrator Version 2 will now use your existing rules and formats as referenced by the MQSIruleng.mpf file.

MQSeries Integrator Version 2 includes a sample message flow which provides the same functionality as offered with MQSeries Integrator V1.1. You must customize this message flow for your installation (first make a copy of the sample, if so desired) by specifying the input queue name on the input node and the names of the failure queue and the noHit queue on the appropriate output nodes. Assign this message flow to the broker(s) where you wish it to run (these brokers must have appropriate access to the database which contains the rules and format definitions). Deploy the configuration changes to the broker(s) and you are now ready to run your existing applications with MQSeries Integrator Version 2.

User exits

If you are migrating from MQ Integrator or MQSeries Integrator V1.0 you will need to rebuild any user exit programs you have written to use the new dynamic linkage mechanism introduced with MQSeries Integrator V1.1 (see "NEON Rules APIs" and "NEON Formatter APIs" for further information).

Enhancing your existing rules

Existing rules and formats defined using the NEONRules and NEONFormatter tools (Version 1.1) will operate without change in the MQSeries Integrator Version 2 environment. The rules and formats may be modified and update by using the existing NEONRules and NEONFormatter GUI tools (included as part of MQSeries Integrator Version 2). The processing capability of such rule sets may also be enhanced by including the rules evaluation as a processing node within a message flow. For example, messages may be warehoused by placing a Warehouse node in the message flow prior to the rules execution. Output messages may be re-routed back to the rules engine for re-evaluation, without the need for going via an intermediate queue. Another example might be to take the output messages from the rules evaluation and send them to the publish/subscribe engine, so that instead of writing to the named queue, the queue manager and queue name is used to form a topic on which to publish the message. Yet another example may take the messages destined for the NoHit queue, add some information into the message header and return the message to the originator.

The next section provides more detailed information regarding the use of the NEONRules and NEONFormatter nodes within a message flow.

Combining the power of NEONRules with MQSI message flow technology

MQSeries Integrator Version 2 allows you to combine the power of the rules and formatter engines of Version 1 with the new flexible message flow model and the extended capabilites provided by the Version 2 message processing nodes.

In designing message flows which combine the use of the NEONRules and NEONFormatter nodes in conjunction with other message processing nodes the following considerations apply:

- a NEONRules or NEONFormatter node can only process messages defined using the NEONFormatter GUI; they cannot process messages defined in the MQSeries Integrator native dictionary using the MQSeries Integrator Version 2 Control Center
- in general, a message processing node can parse a message which has been defined (as an input format) using the NEONFormatter GUI
- in general, a message processing node can not create or modify a message whose format has been defined using the NEONFormatter tool

In practice this means that the following procedures should be followed when implementing message flows which include NEON nodes:

Nodes preceeding NEON node(s) only examine content of message (do not modify the message content in any way)	No action required - message format only needs to be defined as an input format to NEON components.
Nodes following NEON node(s) only examine message properties or message headers (do not examine message body content)	No action required.
Nodes following NEON node(s) examine content of messag (and may also modify content of message).	The format of the message which is passed out of the NEO node must either be defined as both an input and an outpu format in the NEON dictionary or must be defined as an output format in the NEON dictionary and also defined as format in the IBM dictionary. In this second case, the "interchange" message must be a format which can be bui by the NEON formatter and parsed by the IBM parser (suc as a record-oriented format).
Nodes other than NEON node(s) modify message content	The format of the message being passed to the NEON node

	needs to be defined as "shared" format -- that is defined both to NEON and also in the IBM dictionary. This implies that the "interchange" message must be a format which ca be parsed by both the NEON formatter and the IBM parse (such as a record-oriented format or XML).

The easiest way of defining these "interface" messages to both dictionaries is to describe the message format as a COBOL copybook and then inport the record format using both tools. An alternative approach would be to use the NEON XML adapater (available as a separate services offering from NEON), modelling the message(s) using the MQSI Control Center, generating the DTD which represents the message set and importing the DTD through the NEONFormatter GUI.

Choosing an approach for new "message processing applications"

Now that you have installed MQSeries Integrator Version 2 you find yourself faced with choices and decisions. Do you define your message formats using the MQSeries Integrator Control Center or using the NEONFormatter GUI? Should you use the NEON nodes in your message flows or avoid using them because of the associated complexity?

MQSeries Integrator Version 2 provides you with flexibility in making these decisions and the most key decision is probably that of where and how you choose to use the NEONRules node within your message flows. This decision will then, to some extent, dictate where and how you define your message formats, as discussed in the previous section.

One factor which may influence this decision is the type of message format which is supported by each of the message dictionaries. The new MQSeries message dictionary (MRM) is ideally suited to defining message formats for which the "wire format" is either a record-oriented structure (such as might be represented by a C header file or COBOL copybook) or an XML representation. MQSeries Integrator Version 2 includes C and COBOL importers to the MRM as an integral part of the product. A COBOL importer for the NEONFormatter is available as an additional offering from NEON. The NEONFormatter supports messages which are represented as tag-delimited strings (for example SWIFT or FIX messages); these formats are not currently supported by the MRM. Thus choice of the NEONFormatter may be appropriate for those parts of a scenario which require to make use of the SWIFT adapter libraries available from NEON.

The NEONRules engine provides a very convenient way of building a large and efficient multi-way switch for those applications in which a single input message may be transformed and routed to a large number of destinations. Although the same effect can be achieved by chaining together a sequence of Filter nodes, the NEONRules node may provide a convenient option when it is desirable to capture this behaviour within a single node.

Scenarios

In this final section we look at a number of scenarios which use the NEON nodes in different ways within a message flow and consider the steps which need to be taken to implement each of these. These scenarios include:

- warehousing messages before / after rules evaluation
- recursive evaluation of rules
- pub / sub of messages resulting from rules evaluation
- building a message in a message flow to pass to NEONRules node for evaluation
- parsing a message which results from a NEONRules or NEONFormatter transformation

Warehousing of messages processed by the rules engine

We consider two variants on this scenario. In the first instance let us suppose that we wish simply to warehouse every input message prior to passing it to the rules engine for evaluation. We shall assume that the input message formats are defined in the NEONFormatter database. In this case we simply wire a Warehouse node between the MQInput node and the NEONRules node. Configuration of the Warehouse node must be done "manually", that is, any mapping from message fields to database columns must be defined by writing the appropriate SQL SET statements since the Warehouse customiser is not able to read the metadata associated with formats defined in the NEONFormatter meta-database.

Alternatively, consider the case in which we only want to warehouse a subset of the messages which are generated out of the rules evaluation. In this case the criteria for warehousing is that the message was written to a particular output queue. This is achieved by placing a Filter node and a Warehouse node on the main output route from the NEONRules node. The Filter node parses the DestinationList object associated with the message to determine the destination queue name. If the Filter matches then the message is passed to the Warehouse node as well as being passed to the MQOutput node. The Warehouse node simply writes the message as BLOB into the warehouse as it is unable to parse the message. If the output message format is duplicated as input message format for the NEONFormatter then the Warehouse node may extract fields from the message as part of the warehousing operation. However this extraction will still require manual definition (no tree view of the message format in the customiser).

Recursive evaluation of rules

In this scenario it is the case that when a rule fires it may sometimes cause the message to be reformatted and then passed back to the rules engine for further evaluation. In Version 1 this required intermediate queuing on the input queue to the MQSeries Integrator daemon. With Version 2 this intermediate queue can be removed and the re-evaluation can be executed as part of the initial transaction. For this scenario we again assume that the message formats are defined in the NEONFormatter meta-database (and note that in this case they must already be defined as both input and output formats).

A Filter node is wired to the putQueue terminal of the NEONRules node and the filter expression checks whether the destination queue is the same as the input queue for the message flow. The match terminal of the Filter node is wired back to the in terminal of the NEONRules node and the nomatch terminal is wired to the MQOutput node.

No further message definition is required.

Publication of messages generated as a result of the rules evaluation

Suppose that, instead of writing the messages produced as result of evaluating a set of rules directly onto the output queues, you wish to *publish* each message using *QueueManager/QueueName* as a Topic. To achieve this a Compute node is wired to the putQueue terminal of the NEONRules node and then the output of the Compute node is passed to a Publication node. The Compute node parses the DestinationList object associated with the message and uses the information it retrieves to add an RFH2 header to the message specifying the desired Topic. Each message may then be subscribed by multiple applications rather than being passed to a single destination. The subscribing applications may only specify topic-based subscriptions. If you wish also to allow content-based subscriptions then you must ensure that the expected output message formats are also defined as input formats in the NEONFormatter meta-database.

Enriching a message before passing it to a NEONRules node

In this scenario we wish to take a message off the input queue, add some additional information from a reference database and then pass the enriched message to the NEONRules node for evaluation. To do this we connect the MQInput node to a Compute node and connect the Compute node to the NEONRules node. The enriched format of the message must be defined to *both the IBM dictionary and the NEONFormatter meta-database*. This may be achieved in one of two ways: either the message format is defined as a flat COBOL record structure which is then imported into each of the tools or else the message format is defined in the IBM dictionary as an XML message using the Control Center and then the corresponding DTD exported and imported into the NEONFormatter meta-database (which requires the XML adapter which is available as an additional offering from NEON).

The original input format only needs to be defined to either the IBM dictionary (which is preferable since it allows the helpers to be used in the Compute node customiser) or to the NEONFormatter meta-database.