MQSeries® Integrator

System Management Guide

Version 1.0

SC34-5505-00

Note: Before using this information and the product it supports, be sure to read the general information under "Notices" on page 111.

First edition (January 1999)

This edition applies to IBM® MQSeries Integrator, Version 1.0 and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

At the back of this publication is a page titled "Sending your comments to IBM". If you want to make comments, but the methods described are not available to you, please address them to:

IBM United Kingdom Laboratories
Information Development,
Mail Point 095,
Hursley Park,
Winchester,
Hampshire,
England,
SO21 2JN.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Chapter 1
Introduction

The ***MQSeries Integrator System Management Guide*** is for those persons responsible for MQSeries Integrator administration. The System Administrator should have an overall understanding of the MQSeries Integrator product and how it works. It is assumed that the System Administrator is responsible for MQSeries Integrator setup, configuration, and testing. The System Administrator should be supported by a DBA, who administers the databases interacting with MQSeries Integrator, and a network administrator, who ensures that network communications are set up to include MQSeries Integrator.

The information in this guide explains how to set up, run, and test NEONFormatter and NEONRules, and how to configure the MQSeries Integrator Rules daemon.

Product Documentation Set

The MQSeries Integrator documentation set includes:

- ***MQSeries Integrator Installation and Configuration Guide*** helps end users and engineers install and configure MQSeries Integrator.
- ***MQSeries Integrator User's Guide*** helps MQSeries Integrator users understand and apply the program through its graphical user interfaces (GUIs).
- ***MQSeries Integrator System Management Guide*** is for system administrators and database administrators who work with MQSeries Integrator on a day-to-day basis.
- ***MQSeries Integrator Application Development Guide*** assists programmers in writing applications that use MQSeries Integrator APIs.
- ***Programming References*** are intended for users who build and maintain the links between MQSeries Integrator and other applications. The documents include:
  - ***MQSeries Integrator Programming Reference for NEONFormatter*** is a reference to Formatter APIs for those who write applications to translate messages from one format to another.
  - ***MQSeries Integrator Programming Reference for NEONRules*** is a reference to Rules APIs for those who write applications to perform actions based on message contents.

Note

For information on message queuing, refer to the ***IBM MQSeries*** documentation.

Supported Platforms and Compilers

Operating System	DBMS	Compiler
Windows NT 4.0	DB2 5.0 Oracle 7.3 Oracle 8 SQL Server 6.5 Sybase Client 11.1.1 Sybase Server 11.03, 11.5	Microsoft Visual C++ version 4.2
Solaris 2.5.1, 2.6	DB2 5.0 Oracle 7.3 Sybase Client 11.1.1 Sybase Server 11.03, 11.5	Sparcworks C++ compiler version 4.0
HP-UX 10.20	DB2 5.0 Oracle 7.3 Oracle 8 Sybase Client 11.1.1 Sybase Server 11.03, 11.5	HP C++ version 10.34
AIX 4.2	DB2 5.0 Oracle 7.3 Sybase Client 11.1.1 Sybase Server 11.03, 11.5	IBM C Set ++ version 3.1.4

Disk Space and Memory Requirements

Required disk space is a function of the number of queues, formats, and rules. Recommended memory for satisfactory performance depends on message rates, message sizes, and application-specific factors. For Windows NT/SQLServer, the recommended memory is 128 MB; for other platforms, the recommended memory is 256 MB.

Operating System	DBMS	Libraries & Executables
AIX 4.2	DB2 5.0 Oracle 7.3 Sybase 11.03 Sybase 11.5	144 MB 117 MB 130 MB 130 MB
HP-UX 10.20	DB2 5.0 Oracle 7.3 Oracle 8 Sybase 11.03 Sybase 11.5	169 MB 117 MB 117 MB 120 MB 120 MB
Solaris 2.5.1, 2.6	DB2 5.0 Oracle 7.3 Sybase 11.03 Sybase 11.5	166 MB 117 MB 120 MB 120 MB
Windows NT 4.0	DB2 5.0 Oracle 7.3 Oracle 8 SQLServer 6.5 Sybase 11.03 Sybase 11.5	125 MB 117 MB 117 MB 120 MB 120 MB 120 MB

Chapter 2

MQSeries Integrator Overview

MQSeries Integrator provides the flexibility and scalability that allows true application integration. MQSeries Integrator consists of four components:

- MQSeries
- NEONFormatter
- NEONRules
- MQSeries Integrator Rules daemon

MQSeries

MQSeries is message-oriented middleware that is ideal for high-value message handling and high-volume applications because it guarantees each message is delivered only once, and it supports transactional messaging. Messages are grouped into units of work and either all or none of the messages in a unit of work are processed. MQSeries coordinates message work with other transaction work, like database updates, so data integrity is always maintained.

NEONFormatter

NEONFormatter translates messages from one format to another.

NEONFormatter handles multiple message format types from multiple data value sources with the ability to convert and parse messages. Messages can be converted from any described format to any other described format (if fields in input data formats are missing, you can set up defaults for those fields on output). When a message is provided as input to Formatter, the message is parsed and data values are returned. Formatter can handle virtually any message format, including fixed (for example, COBOL records), delimited (for example, C null delimited strings), and variable, tagged, delimited, repetitive and recursive formats (for example, S.W.I.F.T. messages).

Defining message formats in Formatter's database is done through the graphical user interface (GUI). The GUI leads you through the definitions of format components, for example, tags, delimiters, and patterns, to the building of complete message definitions.

NEONRules

NEONRules lets you develop rules for managing message destination IDs, receiver locations, expected message formats, and any processes initiated upon message delivery. The creation and dispatch of multiple messages to multiple destinations from a single input message is supported, and different formats and transport methods for each is allowed. The dynamic nature of NEON's Rules Engine means that rules can be effective immediately, staged over time, or delayed, depending on how the reload messages are timed, allowing flexibility in rapidly changing environments.

NEONRules can examine the value of any field or group of fields in a message to make its determinations. It can aggregate conditions with the Boolean AND and OR operators without architectural limits as to the number or complexity of the expressions.

Note

For more in depth descriptions of the Formatter and Rules modules, refer to the overviews in Chapter 3, *Formatter* and Chapter 4, *Rules* of the ***MQSeries Integrator User's Guide***.

MQSeries Integrator Rules Daemon

The MQSeries Integrator Rules daemon combines MQSeries, Formatter, and Rules in a generic server process. The MQSeries Integrator Rules daemon processes messages from an MQSeries input queue, uses Formatter to parse messages, uses Rules to determine what transformations to perform and where to route the messages, and then puts the output messages on MQSeries queues for delivery to applications.

Chapter 3

Formatter

What is Formatter?

NEONFormatter is packaged as a library of C++ objects that have public functions that constitute the Application Programming Interface (API) or Software Development Kit (SDK). Application developers develop applications that invoke public Formatter functions to parse and reformat messages.

Formatter has two main functions: parsing and reformatting.

- Parse means to parse an input message into individual fields.
- Reformat means to transform an input message into an output message with a different format.

Formatter uses format definitions that describe how to parse an input message and how to format an output message. Format definition data resides in a relational database. Users build and modify format definitions using one of two methods: the Formatter GUI tool or the Formatter management API functions.

The Formatter GUI tool is a program with a graphical user interface that allows users to populate screens with format definition data and store the information in a relational database.

Formatter management API functions are a set of C functions that create format definition data in a relational database. Users can write their own applications that call the management API functions to build format definitions.

Two executables, apitest and msgtest, are delivered with Formatter. These two executables show how to invoke the public functions and serve as tools for validating format definitions.The apitest executable parses an input message and displays a hierarchical representation of the parse tree. The msgtest executable reformats an input message into an output message.

NEONFormatter Consistency Checker checks the correctness of the format definition data in the relational database. As users build and maintain format definition data, they should run the consistency checker periodically to insure the integrity of their data.

The NNFie tool is a command line tool that allows the user to export format definitions from a database to an export file, and to import from the export file into a database. NNFie can import data from a MQSeries Integrator 1.0 export file into a MQSeries Integrator 1.0 database. NNFie exports data from a 1.0 database only.

The Formatter GUI tool has its own import/export function as well. This function uses an export file with a format different from the one used by NNFie.

Fields and Input Controls

Information contained within a structured input message can be broken into individual fields using input controls. Input controls define how to parse an individual field. Defined by a unique name and control information used to define their beginning and end (input control), fields are cohesive parts of a message representing some type of information.

Each field has an associated parse control describing how to identify the field in the message. Input control information includes the data type for the field, tags preceding and/or following the field, the length of the field, the number of times the field repeats within a message, and literals. Repetition count indicates how many times a certain field will appear in a message.

Formatter supports several data types including ASCII String, ASCII Numeric, and Binary. See *Data Types* on page 107 for a complete list of supported data types for this release.

Tags are sets of bits or characters explicitly defining a string of data. For example, <DATE> and </DATE> might mark the beginning and end of a date field in a message.

Literals are symbols used in programming languages such as numbers or strings providing an actual value instead of representing possible values. Literals may only contain ASCII values and are often used as delimiters to separate fields in a message.

Regular Expressions (REs) are strings expressing rules for string pattern matching. Within input parse controls, you can use REs to match ASCII field data in input fields. Instead of searching for a defined literal, you can use a RE to search for complex string patterns in field data. String-matching capabilities implemented comply with the POSIX 1003.2 standard for regular expressions.

Note

For more information on literals and regular expressions, refer to the ***MQSeries Integrator Programming Reference for NEONFormatter***.

Output Controls

For each field in an input message you want to appear in an output message or use to affect a resulting field in an output message, you must have a matching output format control. Output controls specify how to get a starting value for the output field, what data type transformation to perform, and what formatting operations to perform (for example, prefix, suffix, trim).

Defined in much the same way as parse controls, output controls contain additional information such as the type of mathematical operation, prefix and suffix data, user exit routine, pad characters, and default value.

Formats

Simple formats are defined by grouping fields (and their parse or output format controls). Messages are described to Formatter using individual data fields. However, there can be several layers of complexity in a format definition before the actual field values within a message can be determined.

Formats may be one of two types: flat or compound. Flat formats only contain fields and their input or output format controls. Compound formats contain one or more formats, each of which can be either flat or compound.

Input formats (flat or compound) contain fields and their parse controls and are used to parse messages so they can be reformatted according to output formats (flat or compound).

Each format must be defined by the user. However, once a format is defined, the format is available to be used during translation. You can use either the Formatter GUI or Formatter Management APIs to define and configure format descriptions.

Using Reformat(), Formatter can translate a message into a different message using the descriptions for the input and output formats defined by the user. During translation, Formatter uses parse() to break the message into individual fields.

Format Storage

Formatter uses user-defined format descriptions to recognize and parse input messages and reformat output messages. Formatter uses these descriptions to interpret the values in incoming messages and to construct outgoing messages.

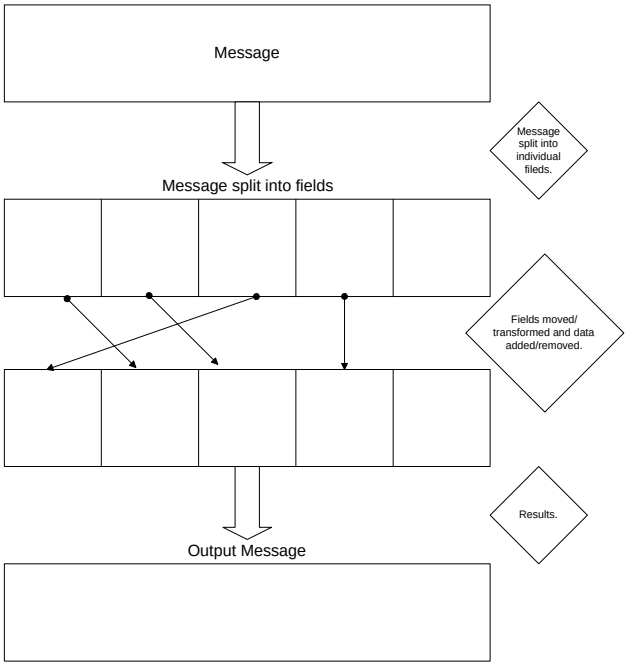
Possible transformations Formatter can handle include:

- Adding, removing, or rearranging data, literals, tags, and delimiters (delimiters are logically cohesive sequences of characters forming a field terminator or format terminator)
- Converting between data types
- Inserting literals into output
- Inserting headers and trailers (including control characters) around any field
- Performing arithmetic operations on numeric data
- Executing user-written data translations functions
- Executing user-written callback functions for user-defined type input field validation and other purposes

Parsing and Reformatting

Formatter can parse a message (using Formatter::Parse()), breaking a message down into its individual fields specified in its input control. When a message is parsed, the intermediate field results can be used.

Or, the parsed message can then be reformatted (using Formatter::Reformat()) in a specified output message format. If the message provided to Reformat() has not been pre-parsed using Parse(), Reformat() calls Parse() before reformatting the message.



Message Formatting

Formatter Configuration

The sqlsvses.cfg file contains information used by Rules and Formatter.

Note

MQSeries Integrator does not use sqlsvses.cfg. The MQSeries Integrator Rules daemon uses a parameter file called MQSIruleng.mpf. However, test programs do use sqlsvses.cfg.

For more information on MQSIruleng, refer to *MQSIruleng* on page 75.

The sqlsvses.cfg File

The sqlsvses.cfg file is the default configuration files and contains information about the database and database server used for MQSeries Integrator executables. This file is created automatically when the libraries are installed and is located in the /bin subdirectory created during the installation process. The password information in the sqlsvses.cfg file is exposed. An alternative is to use the sqlsvses.crypt files.

Note

The sqlsvses.cfg file must be in the same directory as an application using MQSeries Integrator components.

sqlsvses.cfg Parameters

Parameter	Description
session name	Database session name to be used by MQSeries Integrator executables or daemons. This can be any string as long as it is unique within the file.
server name	Server where the MQSeries Integrator database resides.
user name (user id)	Database user name.
password	Database password.
database name	Database name where the MQSeries Integrator tables reside (if applicable). This is not used for Oracle.

Note

The character length for the parameters in the sqlsvses.cfg file is dependent on your server platform and operating system. Line size in the sqlsvses.cfg file is limited to 1024 bytes. Each parameter is separated by a colon.

Encrypting the sqlsvses.cfg file

To use the encryption version of sqlsvses.cfg, run the NNCryptCfg executable against the current sqlsvses.cfg file. A sqlsvses.crypt file is generated. The sqlsvses.crypt file is searched for first. If both a .cfg file and a .crypt file exist in the same directory, the .crypt file is used.

Modifying the location of the sqlsvses file

The default location of the sqlsvses file is the local directory where the executable is invoked. However, the location can be modified and centralized to another location by setting an environment variable.

Set an environment variable (NN_CONFIG_FILE_PATH) to look for the encrypted file. The file name is sqlsvses.crypt, and the default configuration file is not sqlsvses.crypt.

One copy of sqlsvses.cfg can be set up for all directories to point to, eliminating the need for the file in every directory. For example:

```
setenv NN_CONFIG_FILE_PATH/home/smith
```

Or for ksh:

```
export NN_CONFIG_FILE_PATH=/home/smith
```

If the sqlsvses.crypt file is not found, then the sqlsvses.cfg file is used. If neither file is found, an error condition occurs.

Editing the sqlsvses.cfg file

To give MQSeries Integrator the database information it needs for configuration, you must edit the sqlsvses.cfg file. This is an ASCII file that can be edited using any text editor that can save the file in ASCII format.

Text lines in the sqlsvses.cfg file must follow this format:

```
<sessionname>:<servername>:<username>:<password>:
<databasename>
```

A sample text line in the sqlsvses.cfg file for SQL Server and Sybase servers is:

```
new_format_demo:demo_server:demo_user:demo_password:demo_
db:
```

For Oracle servers, <databasename> is not necessary. The end colon (:) must be included in the text line, even if the < database name> parameter is not specified. Oracle servers also use instance names instead of server names.

A sample text line in the sqlsvses.cfg file for an Oracle server is:

```
new_format_demo:demo_instance:demo_user:demo_password: :
```

Note

If the <password> parameter is not specified, leave a blank space between <username> and <databasename> or <instancename>.

Implementing changes to the sqlsvses.cfg file

To implement changes to the sqlsvses.cfg file, you must restart any applications using MQSeries Integrator components.

Import/Export Formats

NNFie

NNFie is a command line tool that allows the user to export format definitions from a database to an export file, and to import from the export file into a database. NNFie can import data from a MQSeries Integrator 1.0 export file into a MQSeries Integrator 1.0 database. NNFie exports data only from a MQSeries Integrator 1.0 database. You cannot export formats from one release of Formatter and import them into a different release of Formatter.

Note

File names (including absolute paths) for both import and export must be no longer than 255 characters.

Command Line Options for NNFie

```
NNFie ((-C [<command file name>]|
    (-i <import file name>|-e <export file name>
    [-m <format name>+]
    [-s <session name>]))
```

[ ] represents optional

() represents grouping

| represents XOR

+ represents one or more

<> means replace with user-provided data

Parameters

Name	Mandatory/ Optional	Description
-C [<command file>]	Optional	Alternate command file name; default file is NNFie.cmd. If this option is provided, NNFie reads command line options from a file instead of the command line.
-i [<import file>]	Mandatory for Import	This parameter is required to import data from the named file, and is mutually exclusive with -e.
-e [<export file>]	Mandatory for Export	This parameter is required to import data from the named file, and is mutually exclusive with -i.
-s [<session name>]	Optional	Name of session in sqlsvses.cfg. Defaults to nnfie.

Name	Mandatory/ Optional	Description
-m [<message type>]	Optional	Specifies the message type to export. The default behavior is to export all messages types within the specified application group.

WARNING!

- Command line option -C allows you to put import/export command options in a text file. You should not use names (for example, format name or session name) in the text file. Also, do not escape the new lines using backslashes such as -e <some file name>. Using backslashes in a command line is not recommended
- If FTP is used with ASCII files to transport the f iles, parts of formats may be missing.

Operational Assumptions

- The file system supports long file names and can also accept the command line syntax described here.
- The operating system supports the concept of standard input, standard output, and standard error stream sources and sinks.

Description

Note

NNFie, NNRie (page 59) and sqlsvses.cfg (page 11) must be in the same directory as NNFie (on UNIX, NNFie.sh) or NNFie.3.0 (on UNIX, NNFie.3.0.sh).

The Import/Export Utility (NNFie) is a script run that exports all components of a given format. The export file for NNFie is not interchangeable with the files created by the GUI.

WARNING!

If you are using a case-insensitive database, you cannot name components the same with only a change in case to identify them. For example, you cannot name one format "f1" and another format "F1". In a case-insensitive environment, you must make each item unique using something other than case differences.

If importing components exported from a context-sensitive database into a context-insensitive database, these differences will cause NNFie to fail during

import if a conflict arises between two components named the same with only case differences.

Import Syntax

Case 1: Import a format

```
$ NNFie -i [<file name>] [-s <session name>]
```

Note

If the format fails to import, an error message is generated and NNFie outputs the data to NEONetferr.

Export Syntax

Case 2: Export an entire database

```
$ NNFie -e [<export file name>] [-s <session name>]
```

Case 3: Export a single format

```
$ NNFie -e [<export file name>] [-m <format name>] [-s
<session name>]
```

Case 4: Export more than one format

```
$ NNFie -e [<export file name>] [-m <format name> <format
name> ...] ] [-s <session name>]
```

Note

Exporting conditional branching rules outputs to <export file name> rules.

Remarks

Environmental Dependencies

This utility requires the following:

1. Previously installed, supported RDBMS system.
2. Previously created Rules database schema.
3. Previously created Formatter database schema.

Export requires the following:

1. Formatter/Rules data in the database created via the Formatter/Rules GUI or Formatter/Rules Management APIs.
2. Enough disk space to hold the output file.

Import requires that the target (MQSeries Integrator r.1.0) database has been created.

Troubleshooting Import Failures

If NNFie fails to import from a given export file, view the NNFie.log to determine the cause for import failure. Two types of errors can cause an import to fail:

1. Conflict errors, i.e., data already exists in the database that conflicts with imported data.
2. Non-conflict errors.

Non-Conflict Error Message (not component specific)

This error message should be complete without any specific component information:

ERROR: <error message>

Non-Conflict Error Message for a Specific Formatter Element

This error message contains both formatter component identification and the data that is being imported:

<Formatter element type>

"<name of the Formatter element>": I/E failed!

ERROR: <error message> [(Formatter management error code)]

<profile - contains all data items related to this Formatter element>

By viewing the data, you should be able to determine the incorrect or missing data items, fix the data in the original database being exported, re-export the data, and then import the newly exported data.

Conflict Error Message for a Specific Formatter Element

In this case, the data being imported conflicts with data already existing in the database. View the data and either remove the conflicting data in the destination database, or fix the data in the originating database, re-export the data, and import the newly exported data.

<Formatter element type>

"<name of the Formatter element>": I/E failed!

ERROR: Import item conflicts with existing Formatter element with the same name

<data item tag (e.g., optional indicator)> (existing = <value> | incoming = <value>)

NNFie Error Messages

Code	Error Name	Error Message	Error Explanation	Response
-4001	NNFIEE_FILE_EXISTS	Given file already exists (so will not replace it)	The specified export file name already exists.	Remove the file or specify a different export file name.

Code	Error Name	Error Message	Error Explanation	Response
-4002	NNFIEE_NO_IMPORT_FILE	No import files by the given name exist	The specified import file name already exists.	Check for the existence of the file.
-4003	NNFIEE_FAILED_TO_READ_FROM_IMPORT_FILE	Failed to read from the import file	The file cannot be read.	Check for the existence of the file or possible access problems.
-4004	NNFIEE_FAILED_TO_SEPARATE_INPUT_DATA	Failed to separate and fetch a piece of the input data	The import file has been corrupted.	Restore or recreate the file.
-4005	NNFIEE_BAD_FILE_STREAM	Bad file stream	Unable to obtain the required file stream.	Check for the existence of the import / export file
-4006	NNFIEE_NAME_PROPERTY_CONFLICT	Conflict with the existing Formatter element with the same name	A format component being imported conflicts with an existing component of the same name.	If you import into a populated format database, you can rename the existing component and import again, or change the incoming component name in the source database and re-export.
-4007	NNFIEE_INVALID_IE_MODE	Invalid import/ export mode (valid: EXPORT_BY_NAME, EXPORT_ALL, IMPORT)	An invalid mode has been specified on the command line or in the command file.	Check the arguments passed to NNFie for correctness.
-4008	NNFIEE_ATTEMPTING_TO_REEXPORT	Attempting to re-export an element that has been exported	A component has been defined that references itself.	Remove the circular reference to this component.
-4009	NNFIEE_FAILED_TO_IMPORT_COMPONENTS	Components have not been imported	During import, one or more of the components required did not import. All components that use the failed component will not import.	Determine why the component did not import correctly.

Code	Error Name	Error Message	Error Explanation	Response
-4010	NNFIEE_INVALID_ FORMATTER_ ELEMENT	Invalid Formatter element type	An unknown format component has been found. The file was exported from an unsupported version of MQSeries Integrator or the file is corrupt.	Check the version of MQSeries Integrator on the source machine. Recover or recreate the export file.
-4011	NNFIEE_INVALID_ NNFIE_FILE	Invalid NNFie file - make sure the file was generated by NNFie	The specified file is incompatible.	Recreate or recover the export file.
-4012	NNFIEE_INVALID_ VERSION_NO	Invalid NNFie version number	The version number found in the file is not supported.	Recreate the file using a supported version of MQSeries Integrator.
-4013	NNFIEE_FAILED_TO_ INVENTORY	Failed to add to the I/E inventory	NNFie was unable to register the component as exported or imported.	Rerun the import/ export.
-4014	NNFIEE_NO_ FORMATS_TO_ EXPORT	No formats to export	The format database does not contain any valid formats to export.	Create valid formats.
-4015	NNFIEE_NOTHING_ TO_IMPORT	Nothing to import	The import file does not contain any format information.	Create an export file from a database that contains formats.
-4016	NNFIEE_FAILED_TO_ ENCRYPT	Encryption failed	NNFie was unable to encrypt the export data successfully.	Rerun the export.
-4017	NNFIEE_FAILED_TO_ DECRYPT	Decryption failed	NNFie was unable to decrypt the import file. This is caused by file corruption.	Recreate of recover the export file.
-4018	NNFIEE_NNFIEERR_ ALREADY_EXISTS	NNFieerr already exists	The error file NNFie.err exists.	Remove the file NNFie.err and rerun.
-4019	NNFIEE_IE_FILE_ ALREADY_EXISTS	I/E file already exists	The specified output file already exists.	Use a new export file name or move/ rename the existing export file.

Code	Error Name	Error Message	Error Explanation	Response
-4020	NNFIEE_FAILED_TO_ OPEN_DBMS_ SESSION	Failed to open DBMS session	NNFie was unable to connect to the database specified in the sqlsvses.cfg file.	Check the entry for NNFie or the session name specified with the -s option in the sqlsvses.cfg file for correctness.
-4021	NNFIEE_FAILED_TO_ OPEN_FMGR	Failed to initialize Formatter manager	NNFie was unable to use the Format manager library.	Check the correctness of the installation of MQSeries Integrator.
-4022	NNFIEE_INVALID_ CNTL_TYPE	Invalid control type	An unknown format control has been found. The file was exported from an unsupported version of MQSeries Integrator or the file is corrupt.	Check the version of MQSeries Integrator on the source machine. Recover or recreate the export file.

NNFie Format Error Messages

Code	Error Name	Error Message	Error Explanation	Response
-4201	NNFIEE_GetFormat	GetFormat failed	The flat or compound format was not accessible in the database through the Format Management API NNFMgrGet Format.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4202	NNFIEE_GetFirst Format	GetFirstFormat failed	The first flat or compound format was not accessible in the database through the Format Management API NNFMgrGetFirst Format.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.

Code	Error Name	Error Message	Error Explanation	Response
-4203	NNFIEE_GETNEXT FORMAT	GetNextFormat failed	The next flat or compound format was not accessible in the database through the Format Management API NNFMgrGetNext Format.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4204	NNFIEE_GetFirst FieldFromInput FormatT	GetFirstFieldFrom InputFormat failed	The first field associated with a flat input format was not accessible in the database through the Format Management API NNFMgrGetFirst FieldFromInput Format.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4205	NNFIEE_GetNextField FromInputFormat	GetNextFieldFrom InputFormat failed	The next field associated with a flat input format was not accessible in the database through the Format Management API NNFMgrGetNext FieldFromInput Format.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4206	NNFIEE_GetFirstField FromOutputFormat	GetFirstFieldFrom OutputFormat failed	The first field associated with a flat output format was not accessible in the database through the Format Management API NNFMgrGetFirst FieldFromOutput Format.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4207	NNFIEE_GetNextField FromOutputFormat	GetNextFieldFrom OutputFormat failed	The next field associated with a flat output format was not accessible in the database through the Format Management API NNFMgrGetNext FieldFromOutput Format.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.

Code	Error Name	Error Message	Error Explanation	Response
-4208	NNFIEE_GetFirstChild Format	GetFirstChildFormat failed	The first child format of a compound format was not accessible in the database through the Format Management API NNFMgrGetFirst ChildFormat.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4209	NNFIEE_GetNextChild Format	GetNextChildFormat failed	The next child format of a compound format was not accessible in the database through the Format Management API NNFMgrGetNext ChildFormat.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4210	NNFIEE_GetOutput Control	GetOutputControl failed	The specified output control was not accessible in the database through the Format Management API NNFMgrGet OutputControl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4211	NNFIEE_GetFirst OutputControl	GetFirstOutput Control failed	The first output control was not accessible in the database through the Formatter Management API NNFMgrGetFirst OutputControl.	Use the secondary Format Management API error code to resolve the problem. See the Formatter Management API error codes.
-4212	NNFIEE_GetNext OutputControl	GetNextOutput Control failed	The next output control was not accessible in the database through the Format Management API NNFMgrGetNext OutputControl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.

Code	Error Name	Error Message	Error Explanation	Response
-4213	NNFIEE_GetParse Control	GetParseControl failed	The specified parse/input control was not accessible in the database through the Format Management API NNFMgrGetParse Control.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4214	NNFIEE_GetFirstParse Control	GetFirstParseControl failed	The first parse/ input control was not accessible in the database through the Format Management API NNFMgrGetFirst ParseControl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4215	NNFIEE_GetNextParse Control	GetNextParseControl failed	The next parse/ input control was not accessible in the database through the Format Management API NNFMgrGetNext ParseControl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4216	NNFIEE_GetDelimiter	GetDelimiter failed	The specified delimiter was not accessible in the database through the Format Management API NNFMgrGet Delimiter.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4217	NNFIEE_GetFirst Delimiter	GetFirstDelimiter failed	The first delimiter was not accessible in the database through the Format Management API NNFMgrGetFirstD elimiter.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.

Code	Error Name	Error Message	Error Explanation	Response
-4218	NNFIEE_GetNext Delimiter	GetNextDelimiter failed	The next delimiter was not accessible in the database through the Format Management API NNFMgrGetNext Delimiter.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4219	NNFIEE_GetField	GetField failed	The specified field was not accessible in the database through the Format Management API NNFMgrGetField.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4220	NNFIEE_GetFirstField	GetFirstField failed	The first field was not accessible in the database through the Format Management API NNFMgrGetFirst Field.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4221	NNFIEE_GetNextField	GetNextField failed	The next field was not accessible in the database through the Format Management API NNFMgrGetNext Field.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4222	NNFIEE_Append FormatToFormat	AppendFormatTo Format failed	The attempt to append one flat or compound format into a compound format failed using the Format Management API NNFMgrAppend FormatToFormat.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.

Code	Error Name	Error Message	Error Explanation	Response
-4223	NNFIEE_AppendField ToInputFormat	AppendFieldToInput Format failed	The attempt to append a field to a flat input format failed using the Format Management API NNFMgrAppend FieldToInput Format.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4224	NNFIEE_AppendField ToOutputFormat	AppendFieldTo OutputFormat failed	The attempt to append a field to a flat output format failed using the Format Management API NNFMgrAppend FieldToOutput Format.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4225	NNFIEE_AppendMath Expression	AppendMath Expression failed	The attempt to append a math expression detail entry to an existing math expression control failed using the Format Management API NNFMgrAppend MathExpression.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4226	NNFIEE_Append LookupEntry	AppendLookupEntry failed	The attempt to append a lookup detail entry to an existing lookup control failed using the Format Management API NNFMgrAppend LookupEntry.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4227	NNFIEE_CreateFormat	CreateFormat failed	The attempt to create a new input/ output flat or compound format failed using the Format Management API NNFMgrCreateFor mat.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.

Code	Error Name	Error Message	Error Explanation	Response
-4228	NNFIEE_CreateParse Control	CreateParseControl failed	The attempt to create a new parse/ input control failed using the Format Management API NNFMgrCreate ParseControl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4229	NNFIEE_CreateOutput Control	CreateOutputControl failed	The attempt to create a new output control failed using the Format Management API NNFMgrCreate OutputControl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4230	NNFIEE_Create Delimiter	CreateDelimiter failed	The attempt to create a new delimiter failed using the Format Management API NNFMgrCreate Delimiter.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4231	NNFIEE_CreateField	CreateField failed	The attempt to create a new field failed using the Format Management API NNFMgrCreate Field.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4232	NNFIEE_SERIOUS_ ERROR_POSSIBLY_ DB_RELATED	GetErrorNo returned serious error number	General database error encountered using the Format Management APIs.	See Format Management API error code -2604.
-4233	NNFIEE_GetDataType Name	GetDataTypename failed	The attempt to retrieve the formal name for the data type code failed due to an invalid data type code associated control.	Run the formatter database consistency verification program to verify data type codes.

Code	Error Name	Error Message	Error Explanation	Response
-4234	NNFIEE_GetDataType	GetDataType failed	The attempt to retrieve the data type code associated with the formal data type name failed.	The NNFie import file does not contain the correct formal data type names. The NNFie import file is corrupt or has been exported from a damaged database.
-4235	NNFIEE_GetFirstUser DefinedType	GetFirstUserDefined Type failed	The first user-defined type was not accessible in the database through the Format Management API NNFMgrGetFirst UserDefinedType.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4236	NNFIEE_GetNextUser DefinedType	GetNextUserDefined Type failed	The next user-defined type was not accessible in the database through the Format Management API NNFMgrGetNext UserDefinedType.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4237	NNFIEE_CreateUser DefinedType	CreateUserDefined Type failed	The attempt to create a new user-defined type failed using the Format Management API NNFMgrCreate UserDefinedType.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4238	NNFIEE_GetFirst Literal	GetFirstLiteral failed	The first literal was not accessible in the database through the Format Management API NNFMgrGetFirst Literal.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.

Code	Error Name	Error Message	Error Explanation	Response
-4239	NNFIEE_GetNext Literal	GetNextLiteral failed	The next literal was not accessible in the database through the Format Management API NNFMgrGetNext Literal.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4240	NNFIEE_GetLiteral	GetLiteral failed	The specified literal was not accessible in the database through the Format Management API NNFMgrGet Literal.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4241	NNFIEE_GetFirstOut MstrCntl	GetFirstOutMstrCntl failed	The first output master control was not accessible in the database through the Format Management API NNFMgrGetFirst OutMstrCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4242	NNFIEE_GetFirst DefaultCntl	GetFirstDefaultCntl failed	The first default control was not accessible in the database through the Format Management API NNFMgrGetFirst DefaultCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4243	NNFIEE_GetFirstUser ExitCntl	GetFirstUserExitCntl failed	The first user exit control was not accessible in the database through the Format Management API NNFMgrGetFirst UserExitCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.

Code	Error Name	Error Message	Error Explanation	Response
-4244	NNFIEE_GetFirstPrePostFixCntl	GetFirstPrePostFixCntl failed	The first prefix/postfix control was not accessible in the database through the Format Management API NNFMgrGetFirstPrePostFixCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4245	NNFIEE_GetFirstSegmentFromMathExpCntl	GetFirstSegmentFromMathExpCntl failed	The first segment of the math expression detail control was not accessible in the database through the Format Management API NNFMgrGetFirstSegmentFromMathExpCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4246	NNFIEE_AppendSegmentToMathExpCntl	AppendSegmentToMathExpCntl failed	The attempt to append a math expression detail entry to an existing math expression failed using the Format Management API NNFMgrAppendSegmentMathExpCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4247	NNFIEE_GetFirstSubstituteCntl	GetFirstSubstituteCntl failed	The first substitute control was not accessible in the database through the Format Management API NNFMgrGetFirstSubstituteCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4248	NNFIEE_GetFirstSubStringCntl	GetFirstSubStringCntl failed	The first substring control was not accessible in the database through the Format Management API NNFMgrGetFirstSubStringCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.

Code	Error Name	Error Message	Error Explanation	Response
-4249	NNFIEE_GetFirstTrim Cntl	GetFirstTrimCntl failed	The first trim control was not accessible in the database through the Format Management API NNFMgrGetFirst TrimCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4250	NNFIEE_GetFirst CollectionCntl	GetFirstCollection Cntl failed	The first output collection control was not accessible in the database through the Format Management API NNFMgrGetFirst CollectionCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4251	NNFIEE_AppendCntl ToCollectionCntl	AppendCntlTo CollectionCntl failed	The attempt to append an output operation to an output operation control failed using the Format Management API NNFMgrAppend CntlToCollection Cntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4252	NNFIEE_GetFirstCntl FromCollection	GetFirstCntlFrom Collection failed	The first output operation collection control was not accessible in the database through the Format Management API NNFMgrGetFirst CntlFrom Collection.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4253	NNFIEE_GetFirst LengthCntl	GetFirstLengthCntl failed	The first length control was not accessible in the database through the Format Management API NNFMgrGetFirst LengthCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.

Code	Error Name	Error Message	Error Explanation	Response
-4254	NNFIEE_GetFirstMath ExpCntl	GetFirstMathExpCntl failed	The first math expression control was not accessible in the database through the Format Management API NNFMgrGetFirst MathExpCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4255	NNFIEE_GetNextOut MstrCntl	GetNextOutMstrCntl failed	The next output master control was not accessible in the database through the Format Management API NNFMgrGetNext OutMstrCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4256	NNFIEE_GetOutMstr Cntl	GetOutMstrCntl failed	The specified output master control was not accessible in the database through the Format Management API NNFMgrGetOut MstrCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4257	NNFIEE_GetNext DefaultCntl	GetNextDefaultCntl failed	The next default control was not accessible in the database through the Format Management API NNFMgrGetNext DefaultCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4258	NNFIEE_GetDefault Cntl	GetDefaultCntl failed	The specified default control was not accessible in the database through the Format Management API NNFMgrGet DefaultCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.

Code	Error Name	Error Message	Error Explanation	Response
-4259	NNFIEE_GetNextUserExitCntl	GetNextUserExitCntl failed	The next user exit control was not accessible in the database through the Format Management API NNFMgrGetNextUserExitCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4260	NNFIEE_GetUserExitCntl	GetUserExitCntl failed	The specified user exit control was not accessible in the database through the Format Management API NNFMgrGetUserExitCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4261	NNFIEE_GetNextPrePostFixCntl	GetNextPrePostFixCntl failed	The next prefix/postfix control was not accessible in the database through the Format Management API NNFMgrGetNextPrePostFixCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4262	NNFIEE_GetPrePostFixCntl	GetPrePostFixCntl failed	The specified prefix/postfix control was not accessible in the database through the Format Management API NNFMgrGetPrePostFixCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4263	NNFIEE_GetNextSegmentFromMathExpCntl	GetNextSegmentFromMathExpCntl failed	The next segment of the math expression detail controls was not accessible in the database through the Format Management API NNFMgrGetNextSegmentFromMathExpCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.

Code	Error Name	Error Message	Error Explanation	Response
-4264	NNFIEE_GetNext SubstituteCntl	GetNextSubstitute Cntl failed	The next substitute control was not accessible in the database through the Format Management API NNFMgrGet SubstituteCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4265	NNFIEE_GetSubstitute Cntl	GetSubstituteCntl failed	The specified substitute control was not accessible in the database through the Format Management API NNFMgrGet SubstituteCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4266	NNFIEE_GetNext SubStringCntl	GetNextSubString Cntl failed	The next substring control was not accessible in the database through the Format Management API NNFMgrGetNext SubStringCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4267	NNFIEE_GetSubString Cntl	GetSubStringCntl failed	The specified substring control was not accessible in the database through the Format Management API NNFMgrGet SubStringCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4268	NNFIEE_GetNextTrim Cntl	GetNextTrimCntl failed	The next trim control was not accessible in the database through the Format Management API NNFMgrGetNext TrimCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.

Code	Error Name	Error Message	Error Explanation	Response
-4269	NNFIEE_GetTrimCntl	GetTrimCntl failed	The specified trim control was not accessible in the database through the Format Management API NNFMgrGetTrim Cntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4270	NNFIEE_GetNextCntl FromCollection	GetNextCntlFrom Collection failed	The next output operation collection control was not accessible in the database through the Format Management API NNFMgrGetCntl FromCollection.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4271	NNFIEE_GetNext LengthCntl	GetNextLengthCntl failed	The next length control was not accessible in the database through the Format Management API NNFMgrGetNext LengthCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4272	NNFIEE_GetLength Cntl	GetLengthCntl failed	The specified length control was not accessible in the database through the Format Management API NNFMgrGet LengthCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4273	NNFIEE_GetNextMath ExpCntl	GetNextMathExp Cntl failed	The next math expression control was not accessible in the database through the Formatter Management API NNFMgrGetNext MathExpCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.

Code	Error Name	Error Message	Error Explanation	Response
-4274	NNFIEE_GetMathExpCntl	GetMathExpCntl failed	The specified math expression control was not accessible in the database through the Format Management API NNFMgrGetMathExpCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4275	NNFIEE_GetNextCollectionCntl	GetNextCollectionCntl failed	The next output collection control was not accessible in the database through the Format Management API NNFMgrGetNextCollectionCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4276	NNFIEE_GetCollectionCntl	GetCollectionCntl failed	The specified output collection control was not accessible in the database through the Format Management API NNFMgrGetCollectionCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4277	NNFIEE_GetUserDefinedType	GetUserDefinedType failed	The specified user-defined type was not accessible in the database through the Format Management API NNFMgrGetUserDefinedType.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4278	NNFIEE_GetNextMathExpression	GetNextMathExpression failed	The next math expression was not accessible in the database through the Format Management API NNFMgrGetNextMathExpression.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.

Code	Error Name	Error Message	Error Explanation	Response
-4279	NNFIEE_GetNextLookupEntry	GetNextLookupEntry failed	The next lookup entry control was not accessible in the database through the Format Management API NNFMgrGetNextLookupEntry.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4280	NNFIEE_GetNextEntryFromSubstituteCntl	GetNextEntryFromSubstituteCntl failed	The next substitute field segment from the substitute control was not accessible in the database through the Format Management API NNFMgrGetNextEntryFromSubstituteCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4281	NNFIEE_AppendEntryToSubstituteCntl	AppendEntryToSubstituteCntl failed	The attempt to create a substitute field segment for the substitute control failed using the Format Management API NNFMgrAppendEntryToSubstituteCntl.	Use the secondary Format Management API error code to resolve the problem. See the Format Management API error codes.
-4500		Fatal internal error	Processing could not continue.	See previous error messages for further information.

Testing Formats

Formatter Test Executables

Two testing executables are provided with Formatter: apitest and msgtest. These executables show how to invoke the public functions and serve as tools for validating format definition.

The apitest executable parses an input message and displays a hierarchical representation of the parse tree. Run apitest to validate input formats and to view how Formatter interpreted a message.

The msgtest executable reformats and input message into an output message. Run msgtest to test input and output formats.

The source code for msgtest and apitest are included in the ***MQSeries Integrator Programming Reference for NEONFormatter*** (see msgtest.cpp and apitest.cpp). Refer to this source code for use of the Formatter API functions.

The apitest Executable

The apitest executable outputs the structure and contents of a message parsed by Formatter. The apitest executable does not test output; it focuses on the input and parse aspects of Formatter.

The apitest command line parameters are:

Usage: `apitest[-d[<filename>]]`

```
-d :parse debug on
```

The -d [filename] parameter sets debugging mode to parse for this run of apitest. [filename] specifies an optional file where debug information is written. If [filename] is not specified, debug information is written to the screen (STDOUT).

Using apitest

To run apitest:

1. At the command line prompt, type apitest.
2. At the prompt, `Enter the input file name:`, type the name of the file in this directory that contains the message to be parsed and reformatted.
3. At the prompt, `Enter the input format name:`, type the name of the input format that will be read from the NNF-FMT table in the database identified in the sqlsvses.cfg file.

The msgtest Executable

The msgtest executable uses input and output formats, delimiters, and other control information read from the database to parse and reformat an input message that is read from a file. The information needed by msgtest must be placed in the database using the GUI or an executable that uses Formatter Management APIs.

The msgtest command line parameters are:

Usage: `msgtest[-li][-lo][-if][-nv][-d[<filename>][-dcp][-dcm][-dco]]`

```
-li:   loud input
-lo:   loud output
-lf:   loud formatted value
-nv:   no validation
-d:    debug on (debug parse only if  -dcp and -dcm and
       -dco not specified)
-dcp:  debug parse on
-dcm:  debug map on
-dco:  debug output on
```

The -d [filename] parameter sets debugging mode to parse for this run of msgtest. [filename] specifies an optional file where debug information is written. If [filename] is not specified, debug information is written to the screen (STDOUT).

Using msgtest

To run msgtest:

1. At the command line prompt, type msgtest.
2. At the prompt, `Enter the input file name:`, type the name of the file in this directory that contains the message to be parsed and reformatted.
3. At the prompt, `Enter the output file name:`, type the name of the file that will contain the reformatted message.
4. At the prompt, `Enter the input format name:`, type the name of the input format that will be read from the NNF-FMT table in the database identified in the sqlsvses.cfg file.
5. At the prompt, `Enter the output format name:`, type the name of the output format that will be read from the NNF_FMT table in the database identified in $msgtest<myFormatterTest.txt>.

Tip

To run msgtest more than once using the same information, create a text file.

The following example shows msgtest command line parameters read from a file on UNIX.

```
$ msgtest<myFormatterTest.txt>
```

The myFormatterTest.txt file contains:

ascii_string	<the input file name containing the message>
output_AS1	<the output file name that will contain the translated message>
AS_IF	<the input format to be read from the database>
AS_NA1_OF	<the output format to be read from the database>

Configuration File

Before running Formatter test executables, verify that the sqlsvses.cfg file includes the database name and server name information used to execute this program. This file must also be in the same directory as the executable program.

Note

For Formatter test executables, the session name to be entered in the sqlsvses.cfg file is new_format_demo.

Example

```
new_format_demo:MyServerName:MyUserName:MyPasswordName:
MyDatabaseName
```

Chapter 4
Rules

Rules is a component of MQSeries Integrator. It is dependent on the Formatter to parse messages for evaluation. Rules has two main functions: evaluating messages against a set of rules and reacting to the evaluation results.

- Evaluating messages means Formatter parses the message and then perform comparisons against individual fields.
- Reacting to the evaluation results means to retrieve a list of rules that hit (their evaluation criteria are true), as well as retrieving a list of subscriptions (actions to perform with options used as parameters).

Rules enables you to evaluate a string of data (a message) and react to the evaluation results. The following overview describes Rules components and what types of APIs are available for rule processing.

Rules is packaged as a library of C++ objects that have public functions that constitute the application programming interface (API) or Software Development Kit (SDK). Application developers design applications that invoke these functions to evaluate messages and retrieve the evaluation results.

Rules uses rules definitions that describe how to parse a message using the format parameters (specified in Formatter) against the rules defined for the message. The rules definitions include subscriptions and the actions to perform if the rule hits. Rules definition data resides in a relational database. Users build and modify rule definitions using one of two methods: the Rules GUI tool or Rules Management API functions.

The Rules GUI tool is a program that allows the user to populate screens with rule definition data and store the information in a relational database.

Rules Management API functions are a set of C functions that create rule definition data in a relational database. Users can write their own interfaces that call the Management API functions to build rule definitions.

The major delivered executable for Rules is the MQSeries Integrator Rules daemon (MQSIruleng). The MQSeries Integrator Rules daemon reads messages off a queue, evaluates the messages, and, based on the results, performs the required reformatting and routing.

The following test executables are delivered with Rules:

- MQSIputdata places a message on a queue with the needed queue options for the MQSeries Integrator Rules daemon.
- MQSIgetdata retrieves all messages and options from a queue.
- NNRTrace evaluates a message against a single rule, displaying a verbose view of each part of the evaluation criteria.

The Rules Consistency Checker utility checks the correctness of the rule definition data in the relational database. As rule definition data is built and

maintained, users should run the consistency checker periodically to insure data integrity.

The NNRie tool delivered with Rules is a command line tool that allows the user to export rule definitions from a database to a file, and to import the exported file into a database. NNRie can import from a MQSeries Integrator 1.0 export file into a MQSeries Integrator 1.0 database. NNRie v1.0 exports data only from a 1.0 database.

Application Groups

Application groups are logical divisions of rule sets for different business needs. You can define as many application groups as you need. For instance, you might want rules for the accounting department and the application development department separated into two groups. You could define "Accounting" as one application group, "Application Development" as another, and then associate rules with each group as appropriate.

Message Types

Message types define the layout of a string of data. Each application group can contain several message types, and a message type can be used with more than one application group. Message types are defined by the user. When using Formatter, a message type is the same as an input format name. This format name is used by Formatter to parse input messages for Rules evaluation.

Rules

When users create rules, they give each rule a rule name and associate the rule name with an application group and message type. Each rule is uniquely identified by its application group / message type / rule name triplet.

Each rule must have the following three items defined: evaluation criteria (an expression containing arguments and operators), subscription information (subscriptions, actions, and options), and permission information. Each of these items is described below.

Expressions, Arguments, Boolean Operators, and Rules Operators

An expression (evaluation criteria) consists of a list of fields, associated operators, and associated comparison data (either static values or other fields) connected with Boolean operators. An argument consists of the combination of a field name, Rules comparison operator, and static value or other field name. Field names depend on the message type (input format name) and they are defined using Formatter. Rules comparison operators are already defined within Rules. Field comparisons can be made against static data or other field values. Arguments are linked together with Boolean operators ‘&’ (AND) and ‘|’ (OR) and parentheses can be used to set the evaluation priority. For more information on operators, refer to ***MQSeries Integrator Programming Reference for NEONRules.***

Subscriptions, Actions, and Options

When a rule evaluates to true, it is considered a "hit." If the rule does not evaluate to true, it is considered a "no-hit." When a rule hits, Rules lets you retrieve associated subscriptions to be taken by the application. These subscriptions are the actions or commands, and the associated parameters or options to execute them.

Subscriptions are lists of actions to take when a message evaluates to true. Each rule must have at least one associated subscription. Subscriptions are uniquely identified within an application group / message type pair by a user-defined subscription name. Permissions must be defined for subscriptions in the same way they are for rules. You can define as many subscriptions as you need. Each action within a subscription is defined by action name and need not be unique, since all actions are intended to be executed in sequence. A single subscription can be shared by multiple rules when the same subscription is associated with each of the rules. In this case, the shared subscription would be retrieved only once no matter how many of its rules hit.

An action has a list of one or more associated options. An option consists of an option name-value pair. The user defines all action names and option name-value pairs.

Rules/Subscription Permissions

Rule and Subscription permissions restrict user access to individual complete rules or subscriptions or their components in the Rules database. The rule is uniquely identified by its application group name, message type, and rule name. A complete rule includes everything associated with it, including an expression (arguments) and associated subscriptions. The subscription is uniquely defined by its application group name, message type, and subscription name. A complete subscription includes everything associated with it including its actions and options. Permissions only apply to managing rule and subscription contents, not rule evaluation.

The Rules component (rule or subscription) or subscription owner is the user who created the component. When the rule or subscription is created, owner information is determined by the software. Owners can update their own permissions, create and update the PUBLIC user’s permissions, and change ownership to another user.

Only read and update permissions are implemented. The owner is given both read and update permission by default. All other users are grouped into a public user group named PUBLIC and given read permissions by default.

Note

Owners can change their own permissions at any time from Read to Update and back again, but they must have update permission to change a rule or subscription contents. Read permission cannot be denied.

APIs

Two types of APIs exist for Rules: Rules APIs and Rules Management APIs.

Use Rules APIs to evaluate rules and retrieve subscription, hit, and no-hit information. Before you evaluate a rule, the rule must exist and you must use CreateRulesEngine() to create a VRule object. After that, you can do as many evaluations and subscription retrievals as needed. When you finish, destroy the MQSeries Integrator Rules daemon object using DeleteRuleEngine().

Use Rules Management APIs to maintain rule information. Add, Read, and Update APIs are implemented and available as well as APIs to delete an entire rule or subscription and all associated information.

Rules Configuration

The sqlsvses.cfg file contains information used by Rules and Formatter.

Note

MQSeries Integrator does not use sqlsvses.cfg. The MQSeries Integrator Rules daemon uses a parameter file called MQSIruleng.mpf. However, test programs do use sqlsvses.cfg.

For more information on MQSIruleng, refer to *MQSIruleng* on page 75.

The sqlsvses.cfg File

The sqlsvses.cfg file is the default configuration file and contains information about the database and database server used for MQSeries Integrator executables. This file is created automatically when the libraries are installed and is located in the /bin subdirectory created during the installation process. The password information in the sqlsvses.cfg file is exposed. An alternative is to use the sqlsvses.crypt files.

Note

The sqlsvses.cfg file must be in the same directory as an application using MQSeries Integrator components.

sqlsvses.cfg Parameters

Parameter	Description
session name	Database session name to be used by MQSeries Integrator executables or daemons. This can be any string as long as it is unique within the file.
server name	Server where the MQSeries Integrator database is resident.
user name (user id)	Database user name.
password	Database password.
database name	Database name where the MQSeries Integrator tables are resident (if applicable). This is not used for Oracle.

Note

The character length for the parameters in the sqlsvses.cfg file is dependent on your server platform and operating system. Line size in the sqlsvses.cfg file is limited to 1024 bytes. Each parameter is separated by a colon (:).

Encrypting the sqlsvses.cfg File

To use the encryption version of sqlsvses.cfg, run the NNCryptCfg executable against the current sqlsvses.cfg file. A sqlsvses.crypt file is generated. The sqlsvses.crypt file is searched for first. If both a .cfg file and a .crypt file exist in the same directory, the .crypt file is used.

Modifying the Location of the sqlsvses File

The default location of the sqlsvses file is the local directory where the executable is invoked. However, the location can be modified and centralized to another location by setting an environment variable.

Set an environment variable (NN_CONFIG_FILE_PATH) to look for the encrypted file. The file name is sqlsvses.crypt, and the default configuration file is not sqlsvses.crypt.

One copy of sqlsvses.cfg can be set up for all directories to point to, eliminating the need for the file in every directory. For example:

```
setenv NN_CONFIG_FILE_PATH/home/smith
```

Or for ksh:

```
export NN_CONFIG_FILE_PATH=/home/smith
```

If the sqlsvses.crypt file is not found, then the sqlsvses.cfg file is used. If neither file is found, an error condition occurs.

Editing the sqlsvses.cfg File

To give MQSeries Integrator the database information it needs for configuration, you must edit the sqlsvses.cfg file. This is an ASCII file that can be edited using any text editor that can save the file in ASCII format.

Text lines in the sqlsvses.cfg file must follow this format:

```
<sessionname>:<servername>:<username>:<password>:
<databasename>
```

A sample text line in the sqlsvses.cfg file for SQL Server and Sybase servers is:

```
new_format_demo:demo_server:demo_user:demo_password:
demo_db:
```

For Oracle servers, <databasename> is not necessary. The end colon (:) must be included in the text line, even if the < database name> parameter is not specified. Oracle servers also use instance names instead of server names.

A sample text line in the sqlsvses.cfg file for an Oracle server is:

```
new_format_demo:demo_instance:demo_user:demo_password: :
```

Note

If the <password> parameter is not specified, leave a blank space between <username> and <databasename> or <instancename>.

Implementing Changes to the sqlsvses.cfg File

To implement the changes made to the sqlsvses.cfg file, restart any applications using MQSeries Integrator components for changes to be recognized by the system.

Note

Use the NNCryptCfg utility to encrypt the password in this file.

System Enhancements for Rules

Oracle

Note

To assign permissions to rules, you must create more than one user in your database.

During installation, a role is created for MQSeries Integrator users: NEONET_USER.

To access MQSeries Integrator databases, users must be created and associated with the NEONET_USER role using the following procedures.

Creating Users

After you install MQSeries Integrator, you must create user names or assign MQSeries Integrator user roles in your database. User names identify individual users to the database.

To create users, type the following command:

```
create user USERNAME identified by PASSWORD;
```

USERNAME and PASSWORD are required parameters.

Granting Roles to Users

Users must be given permissions to access the database data. You can either grant permissions to an individual user or create roles with certain permissions and associate users with specific roles. NEONET_USER is a role created by the MQSeries Integrator installation process.

Grant NEONET_USER role access to created users using the grant command. Syntax for grant is as follows:

```
grant NEONET_USER to USERNAME;
```

The NEONET_USER role is granted to the user identified by USERNAME.

Note

For rules permissions, all users must have only one role granted to them and this role must be the same for all users.

Sybase/SQL Server

The following procedures can be used with Sybase or Microsoft SQL Server. The commands are run within the command line program isql. References to SQL Server include both Sybase and SQL Server.

Except for changing passwords, these procedures are limited to either the system administrator or database owner.

Users must have login accounts and a user name in each database they want to access. Adding login accounts, database users, and object permissions can be done by the system administrator, security officer, or database owner. A single person can occupy one or more of these roles. Check with your database administrator for information about your specific environment.

Creating Login Accounts

Login accounts give users access to the SQL Server. They are created by the system administrator or security officer in the isql command line program using the sp_addlogin system procedure. Syntax for sp_addlogin is as follows:

```
sp_addlogin loginName, password [, defdb [, deflang [,
full-name]]]
```

loginName and password are required parameters. defdb is used to specify a default database for the user. deflang is the name of the default language to use. full-name can be used to enter the full name of the user that owns this account.

Login accounts only give access to the SQL Server. To access a database, a login must be assigned to a user name to the databases the user wants to access.

Assigning Users to a Database

To use a database, the server login must be associated with a user name in the database. Users can be added to a database by the database owner (DBO) using the sp_adduser system procedure.

This procedure must be run from the database in which the user is to be added.

The syntax for sp_adduser is as follows:

```
sp_adduser loginName [, nameInDB] [, group]
```

loginName is the user's server login account. The nameInDB parameter is the name for the user in the database. nameInDB defaults to the loginName if it is not specified. group enables the DBO to add the user to an existing group in

the database. If a group is not specified, the user is placed in the default group, PUBLIC.

Note

For rules permissions, all users must be added as users, not as aliases, and they must be members of the same user group.

Defining User Groups

Each user added to the database must be granted permissions to access objects within that database (unless they are the database owner). During installation, a group is created for MQSeries Integrator users: NEONetUser. To access MQSeries Integrator databases, users must be linked to the MQSeries Integrator group.

Users can be added using either the sp_adduser or sp_changegroup system procedures. The syntax for sp_adduser is discussed in the *Assigning Users to a Database* section above.

The syntax for sp_changegroup is as follows:

```
sp_changegroup groupName, userName
```

groupName is the name of the group to which the user will be added. userName is the database user name.

Permissions for Rules and Subscriptions

Note

You must first create users before you grant permissions. For more information on creating users, refer to the *System Enhancements* section on page 48.

NNDBARuleOwnership

Permissions for Rules and Subscriptions should be managed through either the Rules GUI or Rules Management APIs. However, a tool is provided for System Administration. The NNRDBARuleOwnership utility allows the MQSeries Integrator administrator to list and change the ownership of rules and subscriptions. All rules and subscriptions owned by a specific user can be changed to another user. When rule or subscription ownership is changed, the owner's permissions are transferred to the new owner and previous permissions are overwritten.

Syntax

```
NNRDBARuleOwnership
```

Configuration File

Before running this executable, verify that the sqlsvses.cfg file includes the database name and server name information used to execute this program. This file must also be in the same directory as the executable program. To use the NNRDBARuleOwnership utility, edit the sqlsvses.cfg file to include "rules" as the session name parameter so the utility will connect to the Rules database.

Operations

To use the utility, enter NNRDBARuleOwnership at the command line with no parameters. The utility displays:

```
    Function to perform:
    1  List Rules Owned by a Certain Owner
    2  Change All Rules owned by User A to be Owned By
User B
```

```
    3  List Subscriptions owned by a Certain User
    4  Change All Subscriptions Owned by User A to be
Owned by User B
    >
```

To list rules owned by a certain owner, enter 1 at the prompt (shown as >). The utility displays:

```
    User Name for Owner of Rules (All caps for ORACLE)
    >
```

If you select “list rules owned” (number 1 at the prompt), then the utility lists the application group name, message type name and rule name of all rules owned by the specified user. If you select “change all rules” (number 2 at the prompt), then the utility will not display this rule information.

To change rule ownership, enter 2 at the prompt. The utility displays:

```
    User Name for Current Owner for Rules (All caps for
ORACLE)
    >
    User Name for New Owner of Rules (All caps for ORACLE)
    >
```

To list the subscriptions owned by a certain user, enter 3 at the prompt. The utility displays:

```
    User Name for Owner of Subscriptions (All caps for
ORACLE)
```

A list is displayed showing the Application Group, Message Type, and Subscription Name for all the subscriptions owned by the specified user.

To change subscription ownership, enter 4 at the prompt. The utility displays:

```
    User Name of Current Owner of Subscription (All caps
ORACLE)
    User Name for New Owner of Subscription (All caps for
ORACLE)
```

The owner of the subscription is changed.

Examples

The following examples demonstrate uses of NNRDBARuleOwnership.

Case 1: Listing all rules owned by REL30NEON:--

```
$NNRDBARuleOwnership
Function to perform:
  1 List Rules Owned by a Certain User
  2 Change All Rules owned by User A to be Owned By User B
>1
User Name for Owner of Rules (All caps for ORACLE)
>REL30NEON

Application Group: doc1
Message Type: rp
```

```
Rule Name: d1

Application Group: doc1
Message Type: rp
Rule Name: d5

Application Group: doc2
Message Type: m1
Rule Name: d8
```

Case 2: Listing all rules owned by REL30TEST (not a valid user):--

```
$NNRDBARuleOwnership
Function to perform:
  1 List Rules Owned by a Certain User
  2 Change All Rules owned by User A to be Owned By User B
>1
User Name for Owner of Rules (All caps for ORACLE)
>REL30TEST

Error No: -5509
Error Msg: Unable to find user in database
```

Case 3: Listing all rules owned by REL30NEONUSER2 (no rules owned by user):--

```
$NNRDBARuleOwnership
Function to perform:
  1 List Rules Owned by a Certain User
  2 Change All Rules owned by User A to be Owned By User B
>1
User Name for Owner of Rules (All caps for ORACLE)
>REL30NEONUSER2

Error No: -5519
Error Msg: No permissions were found.
```

Case 4: Changing all rules owned by REL30NEON to be owned by REL30NEONUSER2:--

```
$NNRDBARuleOwnership
Function to perform:
  1 List Rules Owned by a Certain User
  2 Change All Rules owned by User A to be Owned By User B
>2
User Name for Current Owner of Rules (All caps for ORACLE)
>REL30NEON
User Name for New Owner of Rules (All caps for ORACLE)
>REL30NEONUSER2
```

Case 5: Listing all rules owned by REL30NEONUSER2 (now rules are owned by user):--

```
$NNRDBARuleOwnership
```

```
Function to perform:
  1 List Rules Owned by a Certain User
  2 Change All Rules owned by User A to be Owned By User B
>1
User Name for Owner of Rules (All caps for ORACLE)
>REL30NEONUSER2

Application Group: doc1
Message Type: rp
Rule Name: d1

Application Group: doc1
Message Type: rp
Rule Name: d5

Application Group: doc2
Message Type: m1
Rule Name: d8
```

Case 6: Changing all rules owned by REL30TEST to be owned by REL30NEON (not a valid user):--

```
$NNRDBARuleOwnership
Function to perform:
  1 List Rules Owned by a Certain User
  2 Change All Rules owned by User A to be Owned By User B
>2
User Name for Current Owner of Rules (All caps for ORACLE)
>REL30TEST
User Name for New Owner of Rules (All caps for ORACLE)
>REL30NEON

Error No: -5509
Error Msg: Unable to find user in database
```

Case 7: Changing all rules owned by REL30NEONUSER2 to be owned by REL30TEST (not a valid user):--

```
$NNRDBARuleOwnership
Function to perform:
  1 List Rules Owned by a Certain User
  2 Change All Rules owned by User A to be Owned By User B
>2
User Name for Current Owner of Rules (All caps for ORACLE)
>REL30NEONUSER2
User Name for New Owner of Rules (All caps for ORACLE)
>REL30TEST

Error No: -5509
Error Msg: Unable to find user in database
```

Case 8: Changing all rules owned by REL30NEON to be owned by REL30NEONUSER2 (no rules owned by current user):--

```
$NNRDBARuleOwnership
Function to perform:
  1 List Rules Owned by a Certain User
  2 Change All Rules owned by User A to be Owned By User B
>2
User Name for Current Owner of Rules (All caps for ORACLE)
>REL30NEON
User Name for New Owner of Rules (All caps for ORACLE)
>REL30NEONUSER2

Error No: -5519
Error Msg: No permissions were found
```

Case 9: Listing all subscriptions owned by REL40USER:--

```
$NNRDBARuleOwnership
Function to perform:
  1 List Rules Owned by a Certain User
  2 Change All Rules owned by User A to be Owned By User B
  3 List Subscriptions Owned by a Certain User
  4 List All subscriptions Owned by User A to be Owned By
User B
99 Quit
>3
User Name for Owner of Subscriptions (All caps for
ORACLE)
>RELNEON

Application Group: a1
Message Type: rp
Subscription Name: s1

Application Group: a1
Message Type: rp
Subscription Name: s2

Application Group: a1
Message Type: rp
Subscription Name: s3
```

Case 10: Listing all subscriptions owned by REL40TEST (not a valid user):--

```
$NNRDBARuleOwnership
Function to perform:
  1 List Rules Owned by a Certain User
  2 Change All Rules Owned by User A to be Owned By User B
  3 List Subscriptions Owned by a Certain User
  4 Change All subscriptions Owned by User A to be Owned
By User B
99 Quit
>3
User Name for Owner of Subscriptions (All caps for
ORACLE)
>REL40TEST
```

```
Error No: -5509
Error Msg: Unable to find user in database
```

Case 11: Listing all subscriptions owned by REL40USER2 (No subscriptions owned by this user):--

```
$NNRDBARuleOwnership
Function to perform:
  1 List Rules Owned by a Certain User
  2 Change All Rules Owned by User A to be Owned By User B
  3 List Subscriptions Owned by a Certain User
  4 Change All subscriptions Owned by User A to be Owned
By User B
99 Quit
>3
User Name for Owner of Subscriptions (All caps for
ORACLE)
>REL40USER2

Error No: -5519
Error Msg: No permissions were found
```

Case 12: Changing all subscriptions owned by REL40USER1 to REL40USER2:--

```
$NNRDBARuleOwnership
Function to perform:
  1 List Rules Owned by a Certain User
  2 Change All Rules Owned by User A to be Owned By User B
  3 List Subscriptions Owned by a Certain User
  4 Change All subscriptions Owned by User A to be Owned
By User B
99 Quit
>4
User Name for Current Owner of Subscriptions (All caps
for ORACLE)
>REL40USER1
User Name for New Owner of Subscriptions (All caps for
ORACLE)
>REL40USER2

Error No: -5519
Error Msg: No permissions were found
```

Case 13: Listing all subscriptions owned by REL40USER2:--

```
$NNRDBARuleOwnership
Function to perform:
  1 List Rules Owned by a Certain User
  2 Change All Rules Owned by User A to be Owned By User B
  3 List Subscriptions Owned by a Certain User
  4 Change All subscriptions Owned by User A to be Owned
By User B
```

```
99 Quit
>3
User Name for Owner of Subscriptions (All caps for
ORACLE)

Application Group: a1
Message Type: rp
Subscription Name: s1

Application Group: a1
Message Type: rp
Subscription Name: s2

Application Group: a1
Message Type: rp
Subscription Name: s3
```

Case 14: Changing all subscriptions owned by REL40USER2 to REL40TEST (not a valid user):--

```
$NNRDBARuleOwnership
Function to perform:
  1 List Rules Owned by a Certain User
  2 Change All Rules Owned by User A to be Owned By User B
  3 List Subscriptions Owned by a Certain User
  4 Change All subscriptions Owned by User A to be Owned
By User B
99 Quit
>4
User Name for Current Owner of Subscriptions (All caps
for ORACLE)
>REL40USER2

User Name for New Owner of Subscriptions (All caps for
ORACLE)
>REL40TEST

Error No: -5509
Error Msg: Unable to find user in database
```

Case 15: Changing all subscriptions owned by REL40USER1 to REL40USER240(no subscriptions owned by current user):--

```
$NNRDBARuleOwnership
Function to performFunction to perform:
  1 List Rules Owned by a Certain User
  2 Change All Rules Owned by User A to be Owned By User B
  3 List Subscriptions Owned by a Certain User
  4 Change All subscriptions Owned by User A to be Owned
By User B
99 Quit
>4
User Name for Current Owner of Subscriptions (All caps
for ORACLE)
```

```
>REL40USER1

User Name for New Owner of Subscriptions (All caps for
ORACLE)
>REL40USER2

Error No: -5519
Error Msg: No permissions were found
```

Error Conditions

For other errors related to reading rules and subscriptions, refer to ***MQSeries Integrator Programming Reference for NEONRules***.

No Rules for Owner:

Error No: -5519

Error Msg: No permissions were found

Invalid User:

Error No: -5509

Error Msg: Unable to find user in database

Import/Export Rules

NNRie

NNRie is a command line tool that allows the user to export rule definitions from a database to a file and to import the exported file into a database. NNRie can import from a MQSeries Integrator 1.0 export file into a MQSeries Integrator 1.0 database. NNRIE v1.0 exports data only from a MQSeries Integrator 1.0 database.

Syntax

```
NNRie ((-C [<command file name>] |
    -V |
    (-i <import file name>| -e <export file name>
    [[[-a <appname> [...]] [-m <msgname>] [...]] [-r
        <rulename>] [...]])
    [-s <session>]
    [-o]
    [-c <database configuration file name>])))
```

Note

If there are no -a, -m, or -r options, export the entire database.

Operational Assumptions

- The file system supports long file names and can accept the command line syntax described here.
- The operating system supports the concept of standard input, standard output, and standard error stream sources and sinks.

Parameters

Name	Mandatory/ Optional	Description
-C [<command file>]	Optional	Alternate command file; default is NNRie.cmd. If this option is provided, NNRie reads command line options from a file instead of a command line. If -C is present, NNRie expects the other parameters to be in the command file named in the same format as the command line, for example, -C [<file name>], -V OR -i with the remaining information.
-V [version]	Optional	Shows program version information only, and does no processing.
-i [<import file>]	Mandatory for Import	Indicates the program should import data from the named file. This parameter is required to import data, and is mutually exclusive with -e. This parameter may be followed by the name of a file that contains the import data. The referenced file must have been created with the NNRie -e option. The default file name is NNRie.exp.
-e [<export file>]	Mandatory for Export	Indicates the program should export to the named file. This parameter is required to export data, and is mutually exclusive with -i. This parameter may be followed by the name of a file to hold the export data. The default file name is NNRie.exp.
-s <session name>	Optional	The session name corresponding to the session identifier in the MQSeries Integrator configuration file (See the -c option below). The default session tag is “nnrie”.
-o (overwrite flag)	Optional	The default behavior is off (do not overwrite). If this parameter is present during export, it indicates to overwrite the export file. If this parameter is present during import, and a rule or subscription defined in the import file already exists in the importing database, the old rule is deleted if the user is the rule owner and overwritten with the new definition. If the user is not the rule owner, an error is noted and the rule is replaced. If not overwriting rules, any rule that cannot be processed because it already exists in the importing database is noted.

Name	Mandatory/ Optional	Description
-c <config file>	Optional	Indicates the name of the MQSeries Integrator configuration file the program should read to load its session data for access to a database. The default configuration file is “sqlsvses.cfg”.
-a <application group>	Optional	Identifies the application group to export. If a value for this parameter is not identified, all application groups are exported. This parameter can be repeated as many times as necessary to define multiple application groups to export.
-m <message type>	Optional	Specifies the message type to export. This parameter also requires the -a parameter to be set. The default behavior is to export all message types within the specified application group. This parameter can be repeated as many times as necessary to define multiple message types within the same application group.
-r <rule name>	Optional	Specifies the name of the rule to export. This parameter also requires the -a and -m parameters to be set. The default behavior is to export all rules within the specified application group and message type. This parameter can be repeated as many times as necessary to define multiple rules within the same application group and message type.

WARNING!

If FTP is used with ASCII files to transport the f iles, parts of formats may be missing.

Import Syntax

Case 1: Import a Rule. `$ NNRie -i [<file name>] [-s <session name>]`

Export Syntax

Case 2: Export an entire database. `$ NNRie.sh -e [<export file name>] [-s <session name>]`

Case 3: Export a single application group. `$ NNRie.sh -e [-a <app group name>]`

Case 4: Export a message type for an application group. `$ NNRie.sh -e [-a <app group name>][-m <msgtype name>]`

Case 5: Export a single rule. `$ NNRie.sh -e [-a <app group name>] [-m <msgtype name>] [-r <rule name>]`

Case 6: Export more than one rule. `$ NNRie.sh -e [-a <app group name>][-m <msgtype name>][-r <rule name> <rule name>...]`

Exporting conditional branching rules through NNFie.sh outputs to <export file name> .rules.

Remarks

If NNRie is entered with no parameters, NNRie shows a brief usage reminder. If the -V parameter is used, only the version and copyright information is displayed.

Note

The semantics of any file name are operating system dependent, and can be specified as a base name, a fully qualified path and file name, or any other legal specification allowed by the operating system or its shell utility. If specified as a simple base name, the program will create or read the file relative to the directory the user is in when the program was executed.

Note

Subscriptions are added to an Application Group / Message Type (Rule Set), and then they can be associated with multiple rules in the same Application Group / Message Type. The rule name is no longer used to identify subscriptions, so data migration may require subscription names to be generated for uniqueness. The user is prompted to generate the new subscription names.

Testing Rules

Rules Test Programs

The MQSIputdata, MQSIgetdata, and ruletest programs are provided for testing the MQSeries Integrator Rules daemon program. In addition, the NNRTrace program is supplied to provide a debugging utility for Rules. These test programs are explained in this section.

MQSIputdata and MQSIgetdata

The putdata program can be used to put data onto a MQSeries Integrator Rules daemon queue in such a way that the daemon can evaluate the message. The getdata program can be used to get or retrieve messages from a MQSeries Integrator Rules daemon output queue.

Note

MQSIputdata and MQSIgetdata can be used with queues that are not related to the MQSeries Integrator Rules daemon.

MQSIputdata

Syntax--

```
MQSIputdata.exe <parmfileName>
```

Note

The .exe extension in the preceding syntax appears only on Windows NT.

Description--

The MQSIputdata process reads a message from a file and puts the message on the queue specified in the parameter file with OPT_APP_GRP and OPT_MSG_TYPE.

This process sets the two options on the message that the MQSeries Integrator Rules daemon expects, specifically the application group and message type.

Operational Assumptions--

- Queue Manager is up and running.
- Queues have been created.

Parameters--

Put Control

Name	Mandatory/ Optional	Description
inputFileName	Optional	inputFileName is the file containing the message data.
queueName	Mandatory	Name of the queue where the message will be put.
queueManagerName	Optional	Name of the queue manager that owns the queue.
maxUserDataLength	Optional	Maximum message size.
messageCount	Mandatory	Number of messages to put.
showStatistics	Optional	Binary value indicating whether or not statistical information should be output. 1 indicates that the message data should be output; 0 indicates that the message data should not be output.

Put Message

Name	Mandatory/ Optional	Description
format	Optional	Populate the format field of the message descriptor with the indicated value.
expiry	Optional	Populate the expiry field of the message descriptor with the indicated value.
persistence	Optional	Populate the persistence field of the message descriptor with the indicated value. Valid values: MQPER_PERSISTENT = 1 MQPER_NOT_PERSISTENT = 0 MQPER_PERSISTENCE_AS_ Q_DEF = 2
messageType	Optional	Populate the messageType field of the message descriptor with the indicated value. Valid values: MQMT_REQUEST = 1 MQMT_REPLY = 2 MQMT_REPORT = 4 MQMT_DATAGRAM = 8

Name	Mandatory/ Optional	Description
includeHeader	Optional	Specifies whether to include the RF header with the inbound message. 1 indicates that the RF header should be included; zero (0) indicates that the RF header should not be included.
dataformat	Optional	Specifies how to populate the MQRFH.format field. This parameter is valid only if includeHeader = 1.

Put Options

Name	Mandatory/ Optional	Description
OPT_APP_GRP	Optional	Application group associated with the message.
OPT_MSG_ TYPE	Optional	Message type associated with the message.

MQSIgetdata

Syntax--

```
MQSIgetdata.exe <paramFileName>
```

Note

The .exe extension in the preceding syntax appears only on Windows NT.

Operational Assumptions--

- A complete and valid installation of MQSeries Integrator must exist prior to running the MQSeries Integrator Rules daemon. The database must also be running in a stable state prior to running the MQSI getdata program.
- The MQSI getdata program expects that the queue name defined in the command line exists, is enabled, and has messages on it.

Parameters--

Get Control

Name	Mandatory/ Optional	Description
outputFileName	Mandatory	Name of the file to contain messages. getdata.output.
queueName	Optional	Name of the queue holding the messages.
queueManagerName	Optional	Name of the queue manager that owns the queue.

Name	Mandatory/ Optional	Description
maxUserDataLength	Optional	Indicates the maximum message size that the application can accept.
messageId	Optional	Identification of the message to get. If this value and the correlID are not defined, the application gets the next available message from the queue. This field uses an encoded hex representation for the messageId.
correlId	Optional	Correlation ID of the message to get. If this value and messageID are not defined, the application gets the next available message from the queue. The correlID field uses an encoded hex representation of a binary value.
messageCount	Mandatory	Maximum number of the messages to get. The application runs until messageCount messages have been taken off the queue or until the queue is empty.
getTimeout	Optional	The getTimeout value indicates the maximum amount of time to wait for a message to arrive before the application reports that a queue is empty. Upon such a report, the application will exit. getTimeout values are measured in milliseconds.
showStatistics	Optional	Shows statistics about messages taken of the queue. 1 indicates that this feature is enabled; zero (0) indicates that this feature is disabled.
outputToFile	Optional	Indicates whether or not an output should be sent to a file. 1 indicates that output should not be sent to the file; zero (0) indicates that the output should be sent to stderr.
showDescriptor	Optional	Indicates whether or not the message descriptor data should be output. 1 indicates that the message descriptor data should be output; 0 indicates that the message descriptor data should not be output.
showData	Optional	Indicates whether or not the message data should be output. 1 indicates that the message data should be output; zero (0) indicates that the message data should not be output.

Name	Mandatory/ Optional	Description
rollback	Optional	Indicates whether or not the messages should be rolled back after the get operation. 1 indicates that the messages should be rolled back; zero (0) indicates that the messages should not be rolled back.

ruletest

The ruletest program reads a message from a file and evaluates the message using the Rules APIs. This test program does not use Formatter to execute subscriptions.

Syntax

```
ruletest -i <input file name> -m < message type> -a
<application group name> [-v] [-?]
```

Description

The ruletest program reads a message from a file and evaluates the message using the application group / message type defined on the command line. After evaluation, subscriptions are retrieved as they would normally be retrieved and output to the screen, but not executed.

This program does not execute subscriptions using Formatter.

Operational Assumptions

- A complete and valid installation of MQSeries Integrator release 1.0 must exist prior to running the MQSeries Integrator Rules daemon. The database must also be running in a stable state prior to running the ruletest process.
- The ruletest program requires a connection to a database containing both rules and formatter data. This data must reside within the same database.
- The ruletest program uses Formatter to evaluate messages only; the ruletest program does not execute actions.
- The ruletest program uses rules for evaluating and retrieving subscriptions.

Parameters

Name	Mandatory/ Optional	Description
-i <input filename>	Mandatory	The input file from which ruletest will read. The file must reside in the directory that the process is run from or the fully qualified path must be provided.

Name	Mandatory/ Optional	Description
-m <message type>	Mandatory	The ruletest program requires this parameter to evaluate rules.
-a <application group>	Mandatory	The ruletest program requires this parameter to evaluate rules.
-v (verbose)	Optional	The ruletest program logs to the screen if this optional parameter is set. The process defaults to no logging.
-? (usage)	Optional	The ruletest program will display all usage parameters.

Configuration File

Before running this executable, verify that the sqlsvses.cfg file includes the database name and server name information used to execute this program. The sqlsvses.cfg file must also be in the same directory as the executable program.

The session name in the sqlsvses.cfg file is used by ruletest to locate the appropriate line from which to retrieve connection data.

Example

```
rules: MyServerName : MyUserName : MyPassword :
MyDataBase
```

Note

Unless otherwise specified, the ruletest program expects a session name of “rules” for rules and formatter data.

ruletest can be executed using two methods:

1. ruletest evaluates the message using the specified application group / message type if the user enters the parameters listed at the command line.
2. In addition, ruletest can be used interactively by providing no command line parameters.

When ruletest is invoked without command line parameters, it prompts the user for the input file name, application group, message type, verbosity, and whether to reload or not. In interactive mode, ruletest loops through the prompt, optional reload, and evaluation steps.

The optional reload step enables the user to choose whether to refresh the rules data from the database before proceeding.

Note

If ruletest is run with no command line parameters, it prompts the user for the required information.

NNRTrace Rules Debugging Utility

NNRTrace is a rules debugging utility for testing rules. This utility evaluates the rule and message associated with the rule. When the utility completes processing, it displays whether the rule will hit A hit indicates that this message would cause the rule to hit. If the rule hits, the active actions that can be performed by the rule are displayed. If no actions exist, the process fails while evaluating the message.

To use NNRTrace, create an input file for the test procedure, or use the getdata rules test program to retrieve the messages to be tested from a queue.

Syntax

```
NNRTrace -i <input file name> -a <application group> -m
<message type -r <rule name> [-s <session name>] [-o
<output file name>] [-v]
```

Configuration File

Before running this executable, first verify that the sqlsvses.cfg file includes the database name and server name information to be used to execute this program. This file must also be in the same directory as the executable program.

Parameters

Name	Mandatory/ Optional	Description
-i <input filename>	Mandatory	The input file from which NNRTrace will read from. The file must reside in the directory that the process is run from or the fully qualified path must be provided.
-a <application group>	Mandatory	The NNRTrace program requires this parameter to specify the rule.
-m <message type>	Mandatory	The NNRTrace program requires this parameter to specify the rule.
-r <rule name>	Mandatory	The NNRTrace program requires this parameter to specify the rule.
-s <rule session name>	Optional	The rules session name corresponding to the session name in the sqlsvses.cfg file. The session name defaults to “rules.”
-o <output filename>	Optional	The output file to which results of the NNRTrace program will be written. The results are written to standard output by default if this parameter is not specified.
-v (verbose)	Optional	The NNRTrace program logs to the screen if this optional parameter is set. The process defaults to no logging.

Note

If NNRTrace is run without any command line parameters, it prompts the user for the required information.

Chapter 5

The MQSeries Integrator Rules Daemon

The MQSeries Integrator Rules daemon is a content-based rules evaluation and routing engine used to move data from one place to another, depending on the contents of the data. The MQSeries Integrator Rules daemon performs rule evaluations against a specified message and attempts to execute actions for rules that evaluate to true. MQSeries Integrator users can define rules using the GUIs (these are explained in *MQSeries Integrator User's Guide)* or by using the Management APIs (these are explained in *MQSeries Integrator Programming Reference for* NEON*Rules)*. Application programmers can use the Rules APIs to interface database calls to execute rules (these functions are also explained in *MQSeries Integrator Programming Reference for* NEON*Rules*).

Configuration Prior to Using MQSeries Integrator Rules Daemon

To successfully execute MQSeries Integrator, there must be a valid MQSeries installation, all MQSeries queues must be created, and parameter files must be created with database information.

To successfully execute the MQSeries Integrator Rules daemon, a complete and valid installation of MQSeries Integrator must exist prior to using Rules. In addition, rules and formats must be entered and saved before using the MQSeries Integrator Rules daemon. Rules and formats are used by the MQSeries Integrator Rules daemon as defined in this section.

Queues

The MQSeries Integrator Rules daemon uses input and output MQSeries queues. Input queues are specified by name in the parameter file. Output queues are: Failure queue, No Hit queue, and any queues specified by any putqueue action. To create the queues, use the appropriate MQSeries commands.

To have a message successfully evaluated by the MQSeries Integrator Rules daemon, the input message should have these two options set:

```
OPT_APP_GRP
OPT_MSG_TYPE
```

If these options are not set, the MQSeries Integrator Rules daemon assigns defaults. The defaults come from the MQSIruleng.mpf file (or the parameter file name specified on the command line at startup).

OPT_APP_GRP assigns the message to an application group and must match the application group name in the Rules GUI. The OPT_MSG_TYPE must match the message type in rule definitions and the input format name in the format definitions. These two message options allow the MQSeries Integrator Rules daemon to evaluate the message against its rules and only its rules. If the options are not set, the evaluation will not occur and failure processing occurs.

Rules

WARNING!

Unless Reload messages are used, the MQSeries Integrator Rules daemon is not dynamic with respect to rule definition (this also includes subscription definition). Reload is not supported in MQSeries Integrator 1.0. Only rules

defined prior to the MQSeries Integrator Rules daemon startup are used. Any rules added or changed after the MQSeries Integrator Rules daemon startup are not used until the Reload message is read.

Formats

WARNING!

All MQSeries Integrator formats associated with any message put onto any input queue must be entered and saved prior to putting that message onto the input queue. All MQSeries Integrator formats needed during a reformat action must be entered and saved prior to starting the MQSeries Integrator Rules daemon.

Note

For information about entering rules and formats, refer to the ***MQSeries Integrator User Guide*** and the ***Programming Reference*** documents. For information on creating queues, refer to the *MQSeries* documentation.

Putqueue

The Putqueue action takes a message and puts it onto a specified destination MQSeries queue and sets the message type option as the message format type specified. The Putqueue action requires a destination MQSeries queue name and a message format type as options. The message format type must exist in the MQSeries Integrator database. The PutQueue does not perform formatting. If using Rules Management APIs to add the Putqueue action, the action name should be *putqueue* with the option names: OPT_TARGET_QUEUE and OPT_MSG_TYPE (with the case as specified).

For the MQSeries Integrator version of MQSeries Integrator, the Rules daemon uses MQSeries to put to the output queues.

WARNING!

If a subscription does not include the putqueue action, messages will not be put onto any queue and can be lost. The Rules Consistency Checker can be run to determine which subscriptions do not have a Putqueue.

Note

While the reformat and putqueue subscription options are the only actions that can be performed by the Rules Engine, the MQSeries Integrator Rules APIs allow any number of actions and associated options. An application programmer can use MQSeries Integrator APIs in conjunction with independently generated code, in order to execute other types of actions. The

size of your database and performance requirements are the only limitations on the MQSeries Integrator Rules APIs.

Using the MQSeries Integrator Rules Daemon

MQSIruleng

Syntax

```
MQSIruleng -p <parameter file name>
```

Parameters

Operations

Name	Mandatory/ Optional	Description
AllocQuantum	Optional	Unit of memory allocation = 2048 bytes (by default)
ExtendQuantum	Optional	Unit of extension of previously allocated memory block = 1024 bytes (by default)
MaxBufferSize	Optional	Hard limit on growth of memory block = 1048576 bytes (by default). This parameter can be changed in your configuration file.

Logging

Name	Mandatory/ Optional	Description
LogFileName	Optional	LogFileName contains the file specification for the daemon log file. By default, log messages are written to stdout.
LogLevel	Optional	Amount of detail entered in the LogFile. Default = 0. Values: 3-log only fatal 2-log errors and fatal errors 1-log warnings, errors, and fatals 0-log informationals, warning, errors, and fatals

Queues

Name	Mandatory/ Optional	Description
QueueManager Name	Mandatory	Name of the local MQSeries Queue Manager.

Name	Mandatory/ Optional	Description
MaxBackout Count	Optional	Indicates the number of replays before the message is sent to a failure queue. This value can be zero (0) to the maximum imposed by MQSeries. Zero (0) is the default value, and indicates that no replay is allowed.
InputQueue Name	Mandatory	Name of queue used by the MQSeries Integrator Rules daemon to process inbound/input messages.
NoHitQueue Name	Mandatory	Name of queue used by the MQSeries Integrator Rules daemon to place messages that do not satisfy any of the defined rules. A NoHitQueueName value must be supplied by the user; no default value.
FailureQueue Name	Mandatory	Name of queue used by the MQSeries Integrator Rules daemon to place a message in the event where a failure occurred. A FailureQueueName value must be supplied by the user; no default value.
DefaultApp Group	Mandatory	Indicates the default application group. A DefaultAppGroup value must be supplied by the user; no default value.
DefaultMsgType	Mandatory	Indicates the message type value. A DefaultMsgType value must be supplied by the user; no default value.

Rules Database Connections

Name	Mandatory/ Optional	Description
Server Name	Mandatory	The name of the server that you want to connect to.
User ID	Mandatory	Your User ID.
Password	Mandatory	Your password.
Database Instance	Mandatory	The name of the database that you want to connect to.
Database Type	Mandatory	Type the number of the database type: SYBASE = 1 MSSQL=2 Oracle= 3 DB2=4 ODBC=5

Remarks

MQSeries Integrator uses parameter files of the following structure:

```
[Group1]
    field 1 = value 1
    field 2 = value 2
    .
    .
    .
[Group2]
    field 1 = value 1
    field 2 = value 2
    .
    .
    .
[Group3]
    field 1 = value 1
    field 2 = value 2
    .
    .
    .
```

Example

```
############################################################
#
# This is the parameter file for MQSIruleng.exe.
#
# Comments must have a number sign(#) in the first
# column.
#
# Names must be separated from an equals sign by white
# space, and the value also must be separated with white
# space. No white space is allowed in the value string
# itself, nor are trailing comments permitted.
#
# Note that any values in this parameter file will
# override defaults established by the daemon!
############################################################


[Queues]

# Parameters related to queues, MQSeries control, and
# rules engine control

# MQSeries queue manager name
QueueManagerName = ???

# retry limit before replayed message is sent to failure
# queue (zero indicates no replays allowed)
MaxBackoutCount = 0
```

```
# these three queue names are mandatory!
InputQueueName   = ???
NoHitQueueName   = ???
FailureQueueName = ???

# rules default application group and message type values
# (mandatory)
DefaultAppGroup  = ???
DefaultMsgType   = ???

[Logging]
# Log file control..."LogFileName" is the file
# specification for the log, and log levels are:
#   3 - log only fatal errors
#   2 - log errors, and fatal errors
#   1 - log warnings, errors, and fatals
#   0 - log informationals, warnings, errors, and fatals
LogFileName = mqsiruleng.log
LogLevel  = 0


[Rules Database Connection]
#
# NEONet connection information for the database (all
# fields mandatory)
# Note that "DatabaseInstance" is not required for use
# with Oracle databases.
#
ServerName        = ???
UserId            = ???
Password          = ???
DatabaseInstance  = ???
#
# DatabaseType is a numeric with these values:
#   SYBASE  = 1
#   MSSQL   = 2
#   ORACLE  = 3
#   DB2     = 4
#   ODBC    = 5
DatabaseType    = ???
#
# end of file!
#
```

MQSeries Integrator Rules Daemon Processing

The MQSeries Integrator Rules daemon is built on top of the Rules, Formatter, and MQSeries APIs and performs the following procedures in this order:

1. Message processing
2. Subscription execution
3. Failure processing

Message Processing

Message processing evaluates the message against the currently defined rule set for the application group / message type pair. Formatter is called to deconstruct (parse) the input message into component parts (fields). Rules then evaluates these fields. If a message is successfully evaluated, subscriptions are executed. (A subscription is a list of actions.)

If a failure occurs when rules are evaluated against a message, the transaction is rolled back and the transaction end is defined. If a failure occurs during message processing, failure processing begins.

Messages are evaluated against active rules only. If there are no active rules in a rule set (application group / message type), the load will fail, and the message will be sent to the Failure Queue. Only active subscriptions are retrieved for hit rules. If there are no active subscriptions in a rule set, the load will fail, and the message will be sent to the Failure Queue. If none of the hit rules have active subscriptions, the first call of the get subscriptions returns nothing, and the message is sent to the No Hit Queue.

Subscription Execution

After a message (field or fields) is successfully evaluated against its rules, all subscriptions associated with those rules that evaluated to true are executed. If a message is successfully evaluated, and no subscriptions are executed, i.e., no rules evaluate to true, the message is routed to the No Hit Queue.

If there is a failure at any time during subscription execution, the transaction is rolled back, and the transaction end is defined. Once this rollback occurs, failure processing begins.

The subscription actions that can be processed within the MQSeries Integrator Rules daemon are Reformat and Put Queue. Other actions defined require users to write their own daemon to process those actions. The MQSeries Integrator Rules daemon does not execute generic actions.

Note

The MQSeries Integrator Rules daemon shuts down under the following conditions:

- Failure Queue inaccessible
- No Hit Queue inaccessible
- Queue Manager shutdown or inaccessible
- Commit or rollback failure
- Internal error (for example, failure to allocate memory)
- Input queue inaccessible (get is disabled)

Reformat

The Reformat action takes a message with an input format and reformats the message to a message adhering to the specified output format. The Reformat action requires both an input and an output format as options. Formatter APIs are called to perform the reformat of messages. If you are using Rules Management APIs to add the Reformat action, the action name should be "reformat" with the option name: "INPUT_FORMAT" and "TARGET_FORMAT" (in uppercase).

Failure Processing

Failure processing occurs when message processing or subscription execution fails. Failure processing also occurs if there are no active rules or subscription for the application group / message type. Failed messages are routed to the MQSeries Integrator Failure Queue specified in this process. Using the MQSeries Integrator Rules daemon, you can write a process to manage the messages in the Failure Queue.

Message Routing

Based on the outcome of the MQSeries Integrator Rules daemon procedures (message processing, subscription execution and failure processing), messages can be routed to the No Hit Queue, Failure Queue, or to queues specified in a Put Queue action.

- If no subscription actions are successfully executed, messages are routed to the No Hit Queue.
- If failures occur at any time during processing, the message is routed to the Failure Queue.
- If errors occur during execution, all errors are routed to the Log File only if logging is specified.

Rules Caching

When users change data within a rule or rule set specified by an Application Group / Message Type pair, they must signal a running Rules Daemon instance to load the changes into memory.

Notification messages are typically empty and have three options: OPT_APP_GRP and OPT_MSG_TYPE set the application group and message type indicating which rule set to reload, and OPT_RELOAD_RULE_SET set to TRUE indicating to the Rules Daemon to reload the specified rule set.

Sending a Reload Message

To send a reload message:

1. Modify the putdata parameter file: MQSIputdata.mpf.
2. Open your editor and go to the Put Option group in the MQSIputdata.mpf parameter file.
3. Add the following line:

   ```
   OPT_RELOAD_RULE_SET = TRUE
   ```

4. Exit the MQSIputdata.mpf parameter file.
5. To send the reload message, type:

   ```
   MQSIputdata -p MQSIputdata.mpf
   ```

Rules Daemon Security

The MQSeries Integrator Rules daemon can publish messages using one of two methods:

1. With the authority of the user who started the daemon
2. With the authority the publisher

If the first method is used, a message is delivered to a queue with the credentials of whoever originally started the daemon. However, using this method, the rules daemon can be used to put messages to queues that the publisher would not ordinarily be able to access because of security reasons.

If the second method, publisher security, is used, the publish operation fails if a messages is put to a queue that the publisher cannot access because of its security. The message is instead sent to the Rules daemon failure queue.

By default, the Rules daemon uses method one, the security of the user who started the daemon (CredentialsEnabled = 0). To enable method two, publisher security, add the following line to the Rules daemon configuration file:

```
CredentialsEnabled = 1
```

Rules Daemon Shutdown

Sending a Shutdown Message

To send a shutdown message:

1. Add the following line to the Put Option group in the MQSIputdata.mpf file:

   ```
   OPT_SHUTDOWN = SHUTDOWN
   ```

2. To invoke putdata to send the shutdown message, type:

   ```
   MQSIputdata -p MQSIputdata.mpf
   ```

Using Ctrl+c to Shut Down Rules

If you run the Rules daemon interactively, you can exit Rules using *Ctrl+c*. If you use *Ctrl+c*, a message is sent to the log, and Rules exits. If *Ctrl+c* is issued during the processing of a message, the message is rolled back, and it will be in the Failure queue when the Rules daemon is started again.

Note

Using *Ctrl+c* is the abrupt way to shut down Rules. It is better to send a shutdown message, disable the rules input queue, or shut down the queue manager to shutdown Rules. MQSeries connections are not shutdown using *Ctrl+c*.

MQSeries Integrator Rules Daemon Error Messages

Key to Message Severity Codes:

0 = Information

1 = Warning

2 = Error

3 = Fatal Error

Code	Message	Message Severity	Response
Informational Messages			
10000	Received the input message with application group: <insert character string> message type: <insert character string>	0	None. This is an information message.
10001	A putqueue action has begun.	0	None. This is an information message.
10002	Putting message to Failure queue.	0	None. This is an information message.
10003	Performing a reformat operation. Input Message Type: <insert character string> Output Message Type: <insert character string>	0	None. This is an information message.
10004	Message put to target queue but not committed yet. Target Queue Name: <insert character string>	0	None. This is an information message.
10005	Publish operation completed successfully.	0	None. This is an information message.
10006	Reformat operation completed successfully. Input Message Type: <insert character string> Output Message Type: <insert character string>	0	None. This is an information message.
10007	Rules evaluation succeeded. Application Group: <insert character string> Message Type: <insert character string>	0	None. This is an information message.
10008	The Rules evaluation yielded a subscription. Application Group: <insert character string> Message Type: <insert character string>	0	None. This is an information message.
10009	Successfully created output message group. Input Message Type: <insert character string> Output Message Type: <insert character string>	0	None. This is an information message.
10010	Putting message to No Hit queue.	0	None. This is an information message.

Code	Message	Message Severity	Response
10011	A putqueue action has completed.	0	None. This is an information message.
10012	A shutdown has been requested.	0	None. This is an information message.
10013	Reload of a rule set completed successfully.	0	None. This is an information message.
10014	A special control message was detected.	0	None. This is an information message.
10015	User requested abort via signal.	0	None. This is an information message.
10016	Buffer extended.	0	None. This is an information message.
Warning Messages			
10020	No subscriptions found for message with application group: <insert character string> message type: <insert character string>	1	Check the daemon's No Hit queue and verify that no subscriptions exist for that message. This condition does not necessarily represent an error.
10021	A reformatted message exceeded the daemon's maximum allowed message buffer size.	1	Increase the MaxBufferSize.
Error Messages			
10030	Failed to create output message group. Input message type: <insert character string> Output message type: <insert character string> Formatter Error # : <insert number> Formatter Error Message: <insert character string>	2	Refer to Formatter documentation for more information on the Formatter error described by this message.
10031	Failed to open target queue. Target queue name: <insert character string> MQSeries condition code: <insert number> MQSeries reason code: <insert number>	2	Verify that the target queue exists. Refer to the MQSeries documentation for more information on the MQSeries error described by this message.

Code	Message	Message Severity	Response
10032	Failed to put message to target queue. Target queue name:<insert character string> MQSeries condition code: <insert number> MQSeries reason code: <insert number>	2	Verify that the target queue exists. Refer to the MQSeries documentation for more information on the MQSeries error described by this message.
10033	Output type missing for subscription <insert number>.	2	Verify that the output format is specified for this subscription.
10034	Input type missing for subscription <insert number>.	2	Verify that the input format is specified for this subscription.
10035	Failed to reformat message. Input message type: <insert character string> Output message type: <insert character string> Formatter Error #: <insert number> Formatter Error Message: <insert character string>	2	Refer to the Formatter documentation for more information on the Formatter error described by this message.
10036	Rules evaluation failed. Application group: <insert character string> Message type: <insert character string> Rules Error #: <insert number> Rules Error Message: <insert character string>	2	Refer to the Rules documentation for more information on the Rules error described by this message.
10037	Target queue not set for putqueue action.	2	Verify that the putqueue action in your Rules subscription has a target queue defined.
10038	Request-reply messages not supported. Message rejected.	2	Determine which application is sending request messages to the MQSeries Integrator Rules daemon and change the message type from request to datagram.

Code	Message	Message Severity	Response
10039	A corrupt message was detected.	2	Determine which application is sending the corrupt message. Verify that the application is using the method outlined in the example programs to construct the messages it sends to the MQSeries Integrator Rules daemon.
10040	Cannot propagate RF header. No RF header on input message.	2	Either change the publishing application to send messages to the MQSeries Integrator Rules daemon with an RF header, or change the subscription so that the MQS_PROPAGATE option is not set.
10041	No options found for subscription. Subscription Action: <insert character string> Subscription ID: <insert number>	2	Verify that the subscription is valid. A subscription that contains a putqueue action should also contain a target queue option. A subscription that contains a reformat action should also contain both an input format option and a target format option.
10042	Unknown action detected. Action Name: <insert character string> Subscription ID: <insert number>	2	Verify that the actions specified for this subscription are valid. Valid actions are putqueue and reformat.
10043	Putqueue action failed: Action Name: <insert character string> Subscription ID: <insert number>	2	Check the log for additional details about this error.

Code	Message	Message Severity	Response
10044	The input message exceeded the backout count. The message will be sent to the failure queue.	2	Check the log for additional details about this error. This error is preceded in the log by another message indicating why the input message was originally backed out.
10045	A message descriptor extension was detected on the input message. The message will be sent to the failure queue.	2	This error is caused by a version 5 MQSeries application sending messages to an MQSeries Integrator Rules daemon built with version 2 libraries. If possible, upgrade the version of your MQSeries Integrator Rules daemon. Otherwise, stop sending version 5 messages to the MQSeries Integrator Rules daemon.
10046	The Publish operation failed.	2	Check the log for additional information about this error.
10047	A corrupt message options segment was detected.	2	Determine which application is sending the corrupt message. Verify that the application is using the method outlined in the example programs to construct the messages it sends to the MQSeries Integrator Rules daemon.
10048	The input message conversion failed.	2	Check the Log file for details.
10049	Reload of rule set failed.	2	Check the Log file for details.
Fatal Error Messages			
10060	Bad condition code detected.	3	Internal error. Contact product support.
10061	Unrecognized output type in failure message. Message: <insert character string>	3	Contact product support.

Code	Message	Message Severity	Response
10062	A Null Log File handle was detected.	3	Internal error. Contact product support.
10063	Failed to put message on failure queue. MQSeries condition code: <insert number> MQSeries reason code: <insert number>	3	Verify that the failure queue exists and is put enabled. Refer to the MQSeries documentation for more information about the MQSeries error described by this message.
10064	Failed to put message on nohit queue. MQSeries condition code: <insert number> MQSeries reason code: <insert number>	3	Verify that the queue exists and is put enabled. Refer to the MQSeries documentation for more information about this MQSeries error.
10065	Failure queue not specified.	3	Verify that a failure queue is defined by the MQSeries Integrator Rules daemon configuration file.
10066	No input queues were specified.	3	Verify that an input queue is defined by the MQSeries Integrator Rules daemon configuration file.
10067	No Hit queue not specified.	3	Verify that a No Hit queue is defined by the MQSeries Integrator Rules daemon configuration file.
10068	Unexpected argument: <insert character string>	3	Verify the syntax of the command used to run the MQSeries Integrator Rules daemon.
10069	Error connecting to Queue Manager. Queue Manager Name: <insert character string> MQSeries Completion Code: <insert number> MQSeries Reason Code: <insert number>	3	Verify that the queue manager is running. Refer to the MQSeries documentation for more information about this MQSeries error.

Code	Message	Message Severity	Response
10070	Queue Manager Shutdown detected.	3	The MQSeries Integrator Rules daemon shuts down in response to a queue manager shutdown. Restart the queue manager and then restart the MQSeries Integrator Rules daemon.
10071	Failed to commit publish operation. MQSeries condition code: <insert number> MQSeries reason code: <insert number>	3	Refer to the MQSeries documentation for more information about this MQSeries error. Correct the problem and restart the MQSeries Integrator Rules daemon.
10072	Failed to rollback publish operation. MQSeries condition code: <insert number> MQSeries reason code: <insert number>	3	Refer to the MQSeries documentation for more information about this MQSeries error. Correct the problem and restart the MQSeries Integrator Rules daemon.
10073	Failed to open rules session.	3	Check the log for additional information about this error. Make sure that the rules database exists and is available to the MQSeries Integrator Rules daemon. Verify that the database user name and password are valid.
10074	Failed to allocate memory. File Name: <insert character string> Line Number: <insert number>	3	Verify that buffer sizes defined by the MQSeries Integrator configuration file do not exceed system limits. Adjust buffer sizes and restart the daemon.

Code	Message	Message Severity	Response
10075	Message exceeded maximum allowed size. Received message size: <insert number> Maximum allowed size: <insert number>	3	Increase the MaxBufferSize parameter in the MQSeries Integrator configuration file to be greater than or equal to the Received message size given in this error and then restart the daemon.
10076	Error opening a queue. Queue Name: <insert character string> MQSeries Completion Code: <insert number> MQSeries Reason Code: <insert number>	3	Refer to the MQSeries documentation for more information about this MQSeries error. Correct the problem and restart the MQSeries Integrator Rules daemon.
10077	Failed to get a message from an input queue. Input queue name: <insert character string> Completion Code: <insert number> Reason Code: <insert number>	3	Refer to the MQSeries documentation for more information about this MQSeries error. Correct the problem and restart the MQSeries Integrator Rules daemon.
10078	Could not create formatter object. Error Number: <insert number>	3	Check the log for additional information. Refer to the Formatter documentation for additional information on causes of this error.
10079	Could not create rules object. Error Number: <insert number>	3	Refer to the rules documentation for additional information on causes of this error.
10080	Could not access parameter file. Parameter file name: <insert character string>	3	Verify that the MQSeries Integrator Rules daemon parameter file exists.
10081	The parameter file has an invalid format. Parameter file name: <insert character string>	3	Verify that the MQSeries Integrator Rules daemon parameter file has the correct format.
10082	Failed to open rules session. Error #: <insert number> Error Message: <insert character string>	3	Diagnose problem based on Error # and Error Message.

Code	Message	Message Severity	Response
10083	Usage: <insert character string> -p filename	3	Check the command line invocation of the MQSeries Integrator Rules daemon.
10084	Default application group and/or message type not specified.	3	Define the DefaultAppGroup and DefaultMsgType parameters in the Rules daemon configuration file.
10085	Failed to complete inquiry.	3	The MQSeries Integrator Daemon was unable to inquire about the character set ID and encoding used by the Queue Manager. Change the security in the Queue Manager object so that the user who started the MQSeries Integrator Daemon has the right to make an inquiry to the Queue Manager about its properties.

Chapter 6
Consistency Checker

MQSeries Integrator Consistency Checker provides a utility to check the consistency of MQSeries Integrator components. The Consistency Checker lists the objects as invalid that are out of synchronization because of a recovery or bad migration (or for any other reason). It checks whether the records have the corresponding features in the database. All formats and rules in an inconsistent state generate a report indicating the problem. There are no checks across databases; only the specified database is checked.

Most of the items checked verify the internal structure of the rules to confirm that they were properly created; however, some checks verify that user-typed data was correctly entered.

Starting the Consistency Checker from a Command Line

The Consistency Checker Command Line is a UNIX/Korn Shell command. You must have either Oracle SQLPlus, Sybase ISQL, or Informix DB Access installed to run the Consistency Checker using UNIX/Korn Shell commands.

To run the Consistency Check in either Rules or Formatter, change to your CD-ROM drive and access the /bin directory on the MQSeries Integrator CD-ROM.

Rules

To run the Consistency Checker for Rules, type the following UNIX/Korn Shell command:

```
rulecc.ksh <user id> <password> <server name> <database name>
```

Note

The database name is not needed for Oracle.

Note

The file rulecc.sql must be in the same directory as rulecc.ksh, and the user must be able to create new files to run the Consistency Checker.

Formatter

To run the Consistency Checker for Formatter, type the following UNIX/Korn Shell command:

```
formatcc.ksh <user id> <password> <server name> <database name>
```

Reports

The Consistency Checker for Formatter and Rules generates a report similar to the following:

Report Title

First Field Name	Second Field Name	nth Field Name
Field Value 1	Field Value 1	Field Value 1
Field Value 2	Field Value 2	Field Value 2

The report title describes the purpose of the report and each row in the data represents one problem of the same type. For example, the Rules Consistency Checker checks message types that are associated with specific application groups to see if the application group actually exists (this is the third report in the rulecc.ksh output). If an application group is missing, the offending message type is output to the "Message Type" field. If there are no problems, only the report title appears (and possibly a message that no rows were found will be printed as well).

Example: Problem Output

The following example shows a test of the Consistency Checker for Rules where data was forcibly corrupted.

Message types referring to nonexistent application group:

Message Type	Message Id	Application Id
CCFlat 1	50	150

Note

For information on *Starting the Consistency Checker Using the GUI on NT*, see *Using MQSeries Integrator.*

MQSNNRputqCC: Rules Putqueue Action Consistency Checker

This utility, used with both NT and UNIX, goes through all putqueue (PutMessage) actions and checks that queue names specified in putqueue actions exist. The utility lists queue names that do not exist.

Use MQSNNRputqCC from the command line on the machine hosting the MQSeries Integrator Rules daemon and MQSeries queue manager

```
MQSNNRputqCC
Rules Putqueue Consistency Checker (MQSNNRputqCC)

usage: MQSNNRputqCC<rules session name> <queue session OR
queue manager name>
```

For IBM MQSeries, the second parameter is the queue manager name.

Consistency Checker Reports: Rules

The Rules Consistency Checker report provides the following information:

Consistency Checker Reports: Rules	Explanations
Message types in Rules that do not match a Format in Formatter	The message type does not correspond to any input format in the Formatter. The format may have been deleted in Formatter. Do not use Rules in this Message Type.
Rules Unique Sequence Generator with no match on Message Type	These message type / application group pairs do not have the capability to generate unique identifiers for new rules, arguments, subscriptions, or actions. It should be okay to use the database as long as those message types are not used.
Message types in Rules that do not match a Format in Formatter	The message type does not correspond to any input format in the Formatter. The format may have been deleted in Formatter. Do not use Rules in this Message Type.
Rules that refer to nonexistent message types	The associated application group / message type pair does not exist for the rule. You cannot access these rules.
Operations that refer to nonexistent message types	The application group / message type pair does not exist for the argument (operation). You cannot access these rules.
Arguments that refer to nonexistent rules	The rule does not exist for the argument. This may cause load failures.
Arguments that refer to nonexistent operators	The operator does not exist for the argument. This will cause evaluation failure.
Arguments that refer to nonexistent operations	The argument's operation does not exist. This may cause load or evaluation failure.
Subscription actions that refer to nonexistent subscriptions	The subscription does not exist for the action. This may cause load failure.
Subscription Master that refers to nonexistent subscriptions in Subscription List	The subscription does not exist in the subscription list. This may cause load failure.
Subscriptions that refer to nonexistent rules	The rule does not exist for the subscription. This may cause load failure.
Subscriptions in the subscription list that refer to nonexistent message types	The message type / application group pair does not exist for the subscription. You cannot access this subscription.

Consistency Checker Reports: Rules	Explanations
Rules that have Argument Count of Zero	A rule must always have at least one argument associated with it. This report identifies any rules that have a zero (0) argument count. This may cause load or evaluation failure.
Number of Arguments in a Rule does not match the Argument Count indicated for the Rule	The arguments listed in the Argument table do not match the number of arguments in the Rule table. This rule will not work correctly.
Subscription Action (Reformat) Input Format does not exist in the Formatter	The input format entered in a reformat action does not match an input format name in the Formatter. This may cause the daemon to fail reformatting a message.
Subscription Action (Reformat) Target Format does not exist in the Formatter Tables	The target format entered in a reformat action does not match an output format name in the Formatter. This may cause the daemon to fail reformatting a message.
Fields Names in Arguments that refer to nonexistent fields in Formatter	A field name was entered in an argument and the field name is not a valid field in the Formatter. Evaluation may fail or not hit.
Field Names in Arguments that refer to nonexistent Flat Fields in Formatter	A field name was entered in an argument and the field name is not a valid field in the flat input format referred to by the Message Type of the rule. Evaluation may fail or not hit. (NOTE: Currently, the Rules Consistency Checker does not check fields in compound formats.)
Field Name2 (Comparison Value) in Arguments that refer to nonexistent fields in Formatter	A field name was entered in an argument as a comparison value and the field name is not a valid field in the Formatter. Evaluation may fail or not hit.
Field Name2 (Comparison Value) in Arguments that refer to nonexistent Flat Fields in Formatter	A field name was entered in an argument as a comparison value and the field name is not a valid field in the flat input format referred to by the Message Type of the rule. Evaluation may fail or not hit. (NOTE: Currently, the Rules Consistency Checker does not check fields in compound formats.)
Subscriptions with No Actions	All subscriptions must have at least one action. This report displays subscriptions with no actions. This may cause evaluation failure.
Rules with No Active Subscriptions	All rules must have at least one subscription. This report displays rules with no subscriptions. This may cause evaluation failure.

Consistency Checker Reports: Rules	Explanations
WARNING: Rules that may not put message on a queue	When running the MQSeries Integrator Rules daemon, subscriptions for rules that hit probably should end with a 'Put Message' action to route the message. This may not be needed if users provide their own daemon and 'generic' actions.
Boolean Operators that refer to nonexistent rules	The rule does not exist for the argument. This may cause load failures.
Arguments that refer to nonexistent Boolean operators	Boolean operator does not exist for the argument. This will cause load failures.
Boolean operators that have an Argument Count of Zero	A Boolean operator must always have at least two (2) child arguments and / or Boolean operators. This may cause load or evaluation failure.
Number of arguments in a Boolean AND Term does not match the Argument Count indicated for the Boolean Operator	A Boolean AND operator needs the same number of children arguments and / or Boolean operators as is indicated. This will cause evaluation to work incorrectly.
Number of Arguments in a Boolean OR Term is Incorrect	If the expression uses OR, it should have a specific argument count of I. A Boolean OR operator needs a certain number of children arguments and / or Boolean operators as is indicated. This will cause evaluation to work incorrectly.
Boolean operators that refer to nonexistent parent Boolean operators	Children Boolean operators must refer to an existing parent Boolean operator. This may cause load or evaluation failure.
Boolean operators that recurse more than 5 times and maybe infinitely	This expression has many nested expressions, which is okay. However, it can also mean that the expression has a circular reference, which will cause the evaluation failure.
Arguments with static values with invalid lengths	The argument length must be between 0 and 64. This situation may cause load failure or it can cause the rule to never evaluate to true.
Permission Checks	
Unique Sequence Generator invalid for Permission Users	The system cannot add additional users for permissions because it cannot generate a unique identifier.
Hierarchy definitions that are not complete for Rule / Subscription Permissions	The hierarchy definitions must be complete during the installation of Rules with Permissions.
Unique Sequence Generator invalid for Rules / Subscription for Permissions	The system cannot add a new Rule / Subscription permission because it cannot generate a unique identifier.

Consistency Checker Reports: Rules	Explanations
Permissions that do not exist in the hierarchy	Rule / Subscription permissions must refer to valid hierarchy information.
Permissions that are not complete	Rule / Subscription permissions must include Node, Application Group, Message Type, and Rule / Subscription Name to be complete.
Permissions that refer to nonexistent Nodes	Rule / Subscription permissions must refer to the current node.
Permissions that refer to nonexistent Application Groups	Rule / Subscription permissions must refer to a valid application group.
Permissions that refer to nonexistent Message Types	Rule / Subscription permissions must refer to a valid message type / format name.
Permissions that refer to nonexistent Rules	Rule permissions must refer to a valid rule name.
Permissions granted to nonexistent Subscriptions	Subscription permissions must refer to a valid subscription name.
Permissions granted to nonexistent Users	Rules permissions need both a valid user and rule subscription to be complete.
Permissions granted to nonexistent Item	Rule / Subscription permissions must refer to a valid item name.
Permission Access and / or Grants that are not valid for Rules	Current valid Rule permission names are: 'Owner,' 'Read,' and 'Update.' Permission values can be: 'Granted' or 'DenyAll.'
Permission Access and / or Grants that are not valid for Subscription	Current valid Subscription permission names are: 'Owner,' 'Read,' and 'Update.' Permission values can be: 'Granted' or 'DenyAll.'
Rules with multiple owners	Rules can only have one owner. If 'PUBLIC' is the rule owner, every user has de facto ownership.
Subscription with multiple owners	Subscription can only have one owner. If 'PUBLIC' is the rule owner, every user has de facto ownership.
Rules with no Owners	Each rule must have a single owner. A rule with 'PUBLIC' as its owner is basically owned by everyone.
Subscription with no Owners	Each Subscription must have a single owner. A subscription with 'PUBLIC' as its owner is basically owned by everyone.
Users that have no NEONet Rules Data Access	Users for permissions must both have access to the database instance and be in the MQSeries Integrator user group (unless your system is set up in a different way).

Consistency Checker Reports: Formatter

The Formatter Consistency Checker report provides the following information:

Consistency Checker Reports: Formatter	Explanations
Input format fields that refer to nonexistent input controls	Choose valid input parse controls for the fields.
Input format fields that refer to nonexistent fields	Choose valid fields to insert into flat input formats.
Input format fields that refer to nonexistent flat formats	A deleted input format has not been properly removed, there should be no impact.
Incomplete input format fields that refer to field NONE and/or input control NONE	Choose fields other than "NONE" to insert into input flat format. Choose input parse controls other than "NONE" for input fields.
Output format fields that refer to nonexistent flat formats	A deleted output format has not been properly removed. There should be no impact.
Output format fields that refer to nonexistent output controls	Choose valid output format controls for the fields.
Output format fields that refer to nonexistent fields	Choose valid fields to insert into flat output formats.
Output format fields that refer to nonexistent input fields	Choose valid mapped input fields to insert into flat output formats.
Incomplete output format fields that refer to field NONE and/or output control NONE	Choose fields other than "NONE" to insert into output flat format. Choose output format controls other than "NONE" for output fields.
Output format fields that refer to nonexistent access modes	Choose valid access modes for fields in flat output formats.
User defined data type name/value pairs that refer to nonexistent input controls	A deleted input user-validation has not been properly removed. There should be no impact.
User defined data type name/value pairs with invalid types	Database integrity is compromised.
Flat formats that refer to nonexistent format delimiters	Choose valid delimiters for flat formats.

Consistency Checker Reports: Formatter	Explanations
Flat formats that refer to nonexistent formats	A deleted flat format has not been deleted correctly. Database integrity may be compromised.
Flat formats that refer to nonexistent decompositions	Choose valid decomposition (ordered or unordered) for flat formats.
Flat formats that refer to nonexistent terminations	Choose valid termination types for flat formats.
Flat input formats that have no fields	Insert at least one field into format.
Flat output formats that have no fields	Insert at least one field into format.
Compound formats that have no component formats	Insert at least one component format into compound format.
Compound format components that refer to nonexistent parent formats	Deletion of compound format may not have occurred successfully. Database integrity may be compromised.
Compound format components that refer to nonexistent repeat delimiters	Choose valid literals for repeat delimiters for component formats.
Compound format components that refer to nonexistent component formats	Choose valid component formats to insert into compound formats.
Compound format components that refer to nonexistent repeat fields	Choose valid fields for "Field contains repeat count" repeat termination.
Input compound format components that refer to nonexistent repeat terminations	Choose valid repeat termination types for component formats.
Output compound format components that refer to nonexistent repeat terminations	Choose valid repeat termination types for component formats.
Input controls that refer to nonexistent length delimiters	Choose valid literals for length delimiters of input parse controls.
Input controls that refer to nonexistent data delimiters	Choose valid literals for data delimiters of input parse controls.
Input controls that refer to nonexistent tag delimiters	Choose valid literals for tag delimiters of input parse controls.
Input controls that refer to nonexistent data types	Choose valid data types for data portion of input parse control.
Input controls that refer to nonexistent tag data types	Choose valid data types for tag portion of input parse control.
Input controls that refer to nonexistent length data types	Choose valid data types for length portion of input parse control.
Input controls that have invalid default date and time format strings	Date and time data type refers to a date / time format string that is not the legitimate default.

Consistency Checker Reports: Formatter	Explanations
Input controls that have invalid default time format strings	Time data type refers to a time format string that is not the legitimate default.
Input controls that have invalid default date format strings	Date data type refers to a time format string that is not the legitimate default.
Input controls that refer to nonexistent custom date/time format strings	Choose valid custom date/time format strings for input parse controls.
Input controls that refer to nonexistent input control types	Choose valid types for input parse controls.
Input controls that refer to nonexistent data termination types	Choose valid data termination types for input parse controls.
Input controls that refer to nonexistent tag termination types	Choose valid tag termination types for input parse controls.
Input controls that refer to nonexistent length termination types	Choose valid length termination types for input parse controls.
Input controls that refer to nonexistent tag or literal values	Choose valid literals for input parse controls that are literals or that have a tag value.
Input controls that refer to nonexistent length locations	Choose valid length locations for input parse controls.
Input controls of data type endian 2 with data lengths not equal to 2	These are fixed length controls that should have a length of 2.
Input controls of data type endian 4 with data lengths not equal to 4	These are fixed length controls that should have a length of 4.
Input controls of length data type endian 2 with length lengths not equal to 2	These are fixed length controls that should have a length of 2.
Input controls of length data type endian 4 with length lengths not equal to 4	These are fixed length controls that should have a length of 4.
Input controls of data type default date and time with data lengths not equal to length of default date and time format string	These are fixed length controls that should have a length equal to the length of the specified format string.
Input controls of data type default time with data lengths not equal to length of default time format string	These are fixed length controls that should have a length equal to the length of the specified format string.
Input controls of data type default date with data lengths not equal to length of default time format string	These are fixed length controls that should have a length equal to the length of the specified format string.
Input controls of data type custom date/time with data lengths not equal to length of custom date/time format string	These are fixed length controls that should have a length equal to the length of the specified format string.

Consistency Checker Reports: Formatter	Explanations
Output controls that refer to nonexistent data types	Choose valid data types for data portion of output format controls.
Output controls that refer to nonexistent tag data types	Choose valid data types for tag portion of output format controls.
Output controls that refer to nonexistent length data types	Choose valid data types for length portion of output format controls.
Output controls that refer to nonexistent calculation operations	Choose valid calculation operations for output format controls.
Output controls that refer to nonexistent output control types	Choose valid types for output format controls.
Output controls that refer to nonexistent output operations	Choose valid output operations for output format controls.
Output controls that refer to nonexistent field comparison values	Choose valid literals for output format controls of type "Input field value =".
Output controls that refer to nonexistent tag or literal values	Choose valid literals for output format controls of type "Literal" or "Data Field Tag Search".
Output controls that have invalid default date and time format strings	Date and time data type refers to a date / time format string that is not the legitimate default.
Output controls that have invalid default time format strings	Time data type refers to a time format string that is not the legitimate default.
Output controls that have invalid default date format strings	Date data type refers to a date format string that is not the legitimate default.
Output controls that refer to nonexistent custom date / time format strings	Custom date / time data type refers to a custom date / time format string that is not the legitimate default.
Output operations that refer to nonexistent operation types	Database integrity is compromised. Delete operation and re-enter it.
Output operations that refer to nonexistent substitute operations	Database integrity is compromised. Delete operation and re-enter it.
Output operations that refer to nonexistent prefix / suffix operations	Database integrity is compromised. Delete operation and re-enter it.
Output operations that refer to nonexistent default operations	Database integrity is compromised. Delete operation and re-enter it.
Output operations that refer to nonexistent length operations	Database integrity is compromised. Delete operation and re-enter it.
Output operations that refer to nonexistent substring operations	Database integrity is compromised. Delete operation and re-enter it.

Consistency Checker Reports: Formatter	Explanations
Output operations that refer to nonexistent case operations	Database integrity is compromised. Delete operation and re-enter it.
Output operations that refer to nonexistent user exit operations	Database integrity is compromised. Delete operation and re-enter it.
Output operations that refer to nonexistent math expression operations	Database integrity is compromised. Delete operation and re-enter it.
Output operations that refer to nonexistent justify operations	Database integrity is compromised. Delete operation and re-enter it.
Output operations that refer to nonexistent collection operations	Database integrity is compromised. Delete collection and re-enter it.
Collection type output operations that have no collection components	Choose at least one component operation to insert into a collection.
Output operations that refer to nonexistent trim operations	Database integrity is compromised. Delete operation and re-enter it.
Substitute operations that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.
Substitute operations that refer to nonexistent input values	Choose valid literals for substitute input value.
Substitute operations that refer to nonexistent output values	Choose valid literals for substitute output value.
Substitute operations that refer to nonexistent output data types	Choose valid data types for substitute output value.
Collection operations that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.
Collection operation components that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.
Code table entries that refer to nonexistent user defined data types	Extraneous data in the database. Database integrity may be compromised.
User defined data types with invalid data type identifiers	Extraneous data in the database. Database integrity may be compromised.
User defined data types that refer to nonexistent data types	Extraneous data in the database. Database integrity may be compromised.
User defined data types that refer to nonexistent native data types	Choose valid base data types for user defined data types.

Consistency Checker Reports: Formatter	Explanations
Trim operations that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.
Trim operations that refer to nonexistent trim characters	Choose valid literals for trim character.
Trim operations that refer to nonexistent trim choices	Choose valid type for trim operation.
Prefix / suffix operations that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.
Prefix / suffix operations that refer to nonexistent prefixes or suffixes	Choose valid literals for prefixes or suffixes.
Prefix / suffix operations that refer to nonexistent prefix / suffix choice	Choose valid choice for prefix / suffix operation.
Exit operations that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.
Default operations that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.
Default operations that refer to nonexistent padding characters	Choose valid literals to use as default.
Length operations that refer to nonexistent output operations	Database integrity is compromised.
Length operations that refer to nonexistent padding characters	Choose valid literals for padding character.
Math expression operations that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.
Math expression operations that refer to nonexistent rounding modes	Choose valid rounding modes for math expressions.
Math expression components that refer to nonexistent math expression operations	Extraneous data in the database. Database integrity may be compromised.
Case operations that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.
Case operations that refer to nonexistent case choices	Choose valid choices for case operations.
Justify operations that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.

Consistency Checker Reports: Formatter	Explanations
Justify operations that refer to nonexistent justify choices	Choose valid choices for justify operations.
Substring operations that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.
Substring operations that have invalid substring parameters	Choose a substring start position >= 0 and a substring length > 0.
Input parse controls with 2-digit year date / time format strings with invalid year cutoff values	Enter a valid year cutoff value (0 to 99 inclusive) for year cutoff value.

Appendix A
Data Types

The following table both lists and describes supported data types.

Data Type Field Value	Description
Not Applicable	No data type is assumed.
String	A string of standard ASCII characters. Note that non-printable characters are valid as long as they are in the ASCII character set. (EBCDIC characters outside the valid ASCII String range are not valid ASCII String characters. During a reformat from ASCII to EBCDIC if a character being converted is not in the EBCDIC character set the conversion results in a EBCDIC space (hexadecimal 40)).
Numeric	A string of standard ASCII numeric characters.
Binary Data	The Binary data type is used to parse any value and transform that value to an ASCII representation of the value internally in the Formatter. The internal representation takes each byte of the input value and converts it to a readable form. An example of this is parsing a byte whose value is (hexadecimal) 0x9C and transforming that to the internal ASCII representation of 9C, which is the hexadecimal value 0x3943. If this value is used in an output format with the output control's data type set to String, the value placed in the message is ASCII 0x9C. If this value is again placed in an output message with the data type Binary, the ASCII value is not printable and occupies one byte with the value of (hexadecimal) 0x9C. Conversely, an input value of ASCII 3B7A parsed with the String data type can be output using the Binary data type. The output value is (hexadecimal) 0x37BA and occupies 2 bytes in the output message. Valid characters that can be converted to Binary from the String data type are 0 through 9 and A through F. All other characters are invalid.
EBCDIC Data	A string of characters encoded using the EBCDIC (Extended Binary Coded Decimal Interchange Code) encoding used on larger IBM computers. During a reformat from EBCDIC to ASCII, if a character being converted is not in the EBCDIC character set, the conversion results in a space (hexadecimal 20).
IBM Packed Integer	Data type on larger IBM computers used to represent integers in compact form. Each byte represents two decimal digits, one in each nibble of the byte. The final nibble is always a hexadecimal F. For example, the number 1234 is stored as a 3-byte value: 01 23 4F (the number pairs show the hexadecimal values of the nibbles of each byte). The number 12345 is stored as a 3-byte value: 12 34 5F. There is no accounting for the sign of a number, so all numbers are assumed to be positive.

Data Type Field Value	Description
IBM Signed Packed Integer	Data type on larger IBM computers used to represent integers in compact form. This data type takes into account the sign (positive or negative) of a number. Each byte represents two decimal digits, one in each nibble of the byte. The final nibble is a hexadecimal C if the number is positive, and a hexadecimal D if the number is negative. An example of how to generate a default value for an IBM Packed Integer is: Data Type: IBM Signed Packed Decimal Default Value: -12345 (default value in ASCII) Data Length: (Null - use the numbers in this field.) The control is optional and there is no corresponding field in the input message, so Formatter uses the default value, converts it to IBM Signed Packed Decimal, and generates the following output: 12 34 5D. Each pair of numbers represents the two nibbles of a byte. The result is three bytes long.
IBM Zoned Integer	Data type on larger IBM computers used to represent integers. Each decimal digit is represented by a byte. The left nibble of the byte is a hexadecimal F. The right nibble is the hexadecimal value of the digit. For example, 1234 is represented as F1 F2 F3 F4 (the number pairs show the hexadecimal values of the nibbles of each byte).
IBM Signed Zoned Integer	Data type on larger IBM computers used to represent integers. Each decimal digit is represented by a byte. The left nibble of each byte, *except* the last byte, is a hexadecimal F. The left nibble of the last byte is a hexadecimal C if the number is positive, and a hexadecimal D if the number is negative. The right nibble of each byte is the hexadecimal value of the digit. For example, 1234 is represented as F1 F2 F3 C4 (the number pairs show the hexadecimal values of the nibbles of each byte). -1234 is represented as F1 F2 F3 D4.
Little Endian 2	Two-byte integer where the bytes are ordered with the rightmost byte being the high order or most significant byte. For example, the hexadecimal number 0x0102 is stored as 02 01 (where the number pairs show the hexadecimal values of the nibbles of a byte).
Little Swap Endian 2	Two-byte integer where the two bytes are swapped with respect to a Little Endian 2 value. For example, the hexadecimal number 0x0102 is stored as 01 02.
Little Endian 4	Four-byte integer where the bytes are ordered with the rightmost byte being the high order or most significant byte. For example, the hexadecimal number 0x01020304 is stored as 04 03 02 01 (where the number pairs show the hexadecimal values of the nibbles of a byte).
Little Swap Endian 4	Four-byte integer where the two bytes of each word are swapped with respect to a Little Endian 4 value. For example, the hexadecimal number 0x01020304 is stored as 03 04 01 02.
Big Endian 2	Two-byte integer where the bytes are ordered with the leftmost byte being the high order or most significant byte. For example, the hexadecimal number 0x0102 is stored as 01 02 (where the number pairs show the hexadecimal values of the nibbles of a byte).
Big Swap Endian 2	Two-byte integer where the two bytes are swapped with respect to a Big Endian 2 value. For example, the hexadecimal number 0x0102 is stored as 02 01.

Data Type Field Value	Description
Big Endian 4	Four-byte integer where the bytes are ordered with the leftmost byte being the high order or most significant byte. For example, the hexadecimal number 0x01020304 is stored as 01 02 03 04 (where the number pairs show the hexadecimal values of the nibbles of a byte).
Big Swap Endian 4	Four-byte integer where the two bytes of each word are swapped with respect to a Big Endian 4 value. For example, the hexadecimal number 0x01020304 is stored as 02 01 04 03.
Decimal, International	Data type where every third number left of the decimal point is preceded by a period. The decimal point is represented by a comma. Numbers right of the decimal point represent a fraction of one unit. For example, the number 12345.678 is represented as 12.345,678. Decimal international data types can contain negative values.
Decimal, U.S.	Data type where every third number left of the decimal point is preceded by a comma. The decimal point is represented by a period. Numbers right of the decimal point represent a fraction of one unit. For example, the number 12345.678 is represented as 12,345.678. Decimal US data types can contain negative values.
Unsigned Little Endian 2	Like Little Endian 2, except that the value is interpreted as an unsigned value.
Unsigned Little Swap Endian 2	Like Little Swap Endian 2, except that the value is interpreted as an unsigned value.
Unsigned Little Endian 4	Like Little Endian 4, except that the value is interpreted as an unsigned value.
Unsigned Little Swap Endian 4	Like Little Swap Endian 4, except that the value is interpreted as an unsigned value.
Unsigned Big Endian 2	Like Big Endian 2, except that the value is interpreted as an unsigned value.
Unsigned Big Swap Endian 2	Like Big Swap Endian 2, except that the value is interpreted as an unsigned value.
Unsigned Big Endian 4	Like Big Endian 4, except that the value is interpreted as an unsigned value.
Unsigned Big Swap Endian 4	Like Big Swap Endian 4, except that the value is interpreted as an unsigned value.
Date and Time*	Based on the international ISO-8601:1988 standard datetime notation: YYYYMMDDhhmmss. See the first paragraph of each of the Date and Time type descriptions for details on representing Date and Time components. Combined dates and times may be represented in any of the following list of base data types. For some data types, a minimum of 8 bytes is required. The list includes: Numeric, String, and EBCDIC.
Time*	Based on the international ISO-8601:1988 standard time notation: hhmmss where hh represents the number of complete hours that have passed since midnight (between 00 and 23), mm is the number of minutes passed since the start of the hour (between 00 and 59), and ss is the number of seconds since the start of the minute (between 00 and 59). Times are represented in 24-hour format. Times may be represented in any of the following list of base data types. For some data types, a minimum of 4 bytes is required. The list includes: Numeric, String, and EBCDIC.

Data Type Field Value	Description
Date*	Based on the international ISO-8601:1988 standard date notation: YYYYMMDD where YYYY represents the year in the usual Gregorian calendar, MM is the month between 01 (January) and 12 (December), and DD is the day of the month with a value between 01 and 31.Dates may be represented in any of the following list of base data types. For some data types, a minimum of 4 bytes is required. The list includes: Numeric, String and EBCDIC.
Custom Date and Time*	Custom Date and Time enables users to specify different formats of dates, times, and combined dates and times. Date / Time formats may include: 1) Variations in year (2- or 4-digit year representation: YY or YYYY). 2) Variations in month –use of a month number (01-12) or three-letter abbreviation (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC). 3) Variations in the day of the month – use of a day of the month number (01-31). 4) Variations in hour – 12-hour or 24-hour representation, with or without a meridian indicator (AM or PM.) 5) Custom date / time formats are available in the Format drop-down list. Custom date / time formats must have a base data type of Numeric, String, or EBCDIC.

Appendix B
Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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:

```
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.
```

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

```
IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan
```

The following paragraph does not apply to the United Kingdom or any other 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 NON-INFRINGEMENT, 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.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this document to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those

Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

```
IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England,
SO21 2JN.
```

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks and Service Marks

The following, which appear in this book or other MQSeries Integrator books, are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

MQSeries
AIX
DB2
IBM

NEONFormatter and NEONRules are trademarks of New Era of Networks, Inc. in the United States and / or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and / or other countries.

UNIX is a registered trademark in the United States and / or other countries licensed exclusively through X / Open Company Limited.

Other company, product, or service names may be the trademarks or service marks of others.

Index

N

O

P

R

S

T

U

Sending your comments to IBM

MQSeries Integrator

System Management Guide

SC34-5505-00

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information in this book only and the way in which the information is presented.

To request additional publications or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By fax:
  - From outside the U.K., use your international access code followed by 44 1962 870229
  - From within the U.K., use 01962 870229

Electronically, use the appropriate network ID:

- IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
- IBMLink: HURSLEY(IDRCF)
- Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The page number or topic number to which your comment applies
- Your name / address / telephone number / fax number / network ID

SC34-5505-00