MQSeries® Integrator

System Management Guide

Version 1.1

SC34-5505-01

Note: Before using this information, and the product it supports, be sure to read the general information under *Notices* on page 265**.**

Second edition (June 1999)
This edition applies to IBM® MQSeries Integrator, Version 1.1 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.

© Copyright New Era of Networks, Inc., 1998, 1999. All rights reserved.

© Copyright International Business Machines Corporation, 1999. All rights reserved.

Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Chapter 1
Introduction

The *MQSeries Integrator* ***System Management*** *Guide* is for those 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 database administrator (DBA), who administers the databases interacting with MQSeries Integrator, and a network administrator, who ensures that network communications are configured 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.
- ***Application Programming Reference*** assists programmers in writing applications that use MQSeries Integrator APIs.

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

Summary of Changes

This document is a major revision in support of the functional changes introduced with Version 1.1. This revision also includes maintenance and editorial changes.

Formatter on page 13 has been updated to include the following new functionality.

- NNFie readable file
- NNFie file header on export file
- NNFie smaller record sizes
- NNFie additional flexibility to resolve component conflicts when importing formats
- NNFie ability to export conditional branching controls
- NNFie inventory export file
- Math Expression enhancements
- Formats can be migrated from an existing to a database to a new database using NNFie.

Rules on page 87 has been updated to include the following new functionality.

- NNRie readable file
- NNRie export/import of orphan subscriptions
- NNRie additional flexibility to resolve component conflicts when importing rules
- NNRie Optional trace (line-by-line status)
- NNRie Inventory export file
- NNRie enhanced error handling
- Tunable parameters as part of putdata.mpf and getdata.mpf. The parameters allow you to customize control and performance of the the putdata and getdata modules.

- Rules can be migrated from an existing to a database to a new database using NNRie.

The MQSeries Integrator Rules Daemon on page 195 has been updated to include the following functionality:

- Definition of multiple input queues, each with its own set of default values.
- Additional tunable parameters as part of the ruleng.mpf file. The parameters allow you to customize control and performance of the MQSI Rules Daemon to your environment.
- Encryption of the UserId and Password keys of the Rules Database Connection parameter group using the MQSIencrypt utility.
- The MQSI Rules daemon can be run as an NT Service.
- Updated MQSI Rules daemon error messages, including information, error, and fatal error messages that may result from use of the tunable parameters for ruleng.mpf, putdata.mpf, and getdata.mpf. The error message codes reflect change to a severity-based numbering scheme.

Supported Platforms and Compilers

Operating System	DBMS	Compiler
AIX 4.2, 4.3	DB2 5.0 DB2 5.2 Oracle 7.3.4 Oracle 8.0.5 Sybase 11.5 Sybase 11.9	IBM C Set ++ version 3 or later
HP-UX 10.20	DB2 5.0 DB2 5.2 Oracle 7.3.4 Oracle 8.0.5 Sybase 11.5 Sybase 11.9	HP C++ version 10.40 (HP-UX 10.20)
Solaris 2.5.1, 2.6	DB2 5.0 DB2 5.2 Oracle 7.3.4 Oracle 8.0.5 Sybase 11.5 Sybase 11.9	Sparcworks C++ compiler version 4.2
Windows NT 4.0	DB2 5.0 DB2 5.2 Oracle 7.3.4 Oracle 8.0.5 SQL Server 6.5 Sybase 11.5 Sybase 11.9	Microsoft Visual C++ version 6.0

Disk Space and Memory Requirements

Required disk space is dependent on 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.

MQSeries Integrator Disk Space Requirements

For Solaris, the /var/tmp file system requires at least 250 MB of free space to unpack the MQSeries and MQSeries Integrator products.

The minimum database allocation requires 20 MB.

MQSeries Integrator binaries require 150 MB.

MQSeries base code and server require a minimum of 25-30 MB of disk space to be available for the product code and data.

MQSeries Integrator documentation requires 1550 MB of disk space (HTML files: 35 MB, PDF files: MB).

The GUI requires 40 MB.

Library and Executable Disk Space Requirements

The following table contains library and executable disk space requirements.

Operating System	DBMS	Libraries & Executables
AIX 4.2, 4.3	DB2 Oracle Sybase	50 MB 115 MB 120 MB
HP-UX 10.20, 11	DB2 Oracle Sybase	80 MB 120 MB 85 MB
Solaris 2.51, 2.6	DB2 Oracle Sybase	90 MB 85 MB 80 MB
Windows NT 4.0	DB2 Oracle SQLServer Sybase	75 MB 60 MB 60 MB 60 MB

Year 2000 Readiness Disclosure

MQSeries Integrator, when used in accordance with its associated documentation, is capable of correctly processing, providing, and/or receiving date information within and between the twentieth and twenty-first centuries, provided that all products (for example, hardware, software, and firmware) used with this IBM program properly exchange accurate date information with it.

Customers should contact third party owners or vendors regarding the readiness status of their products.

IBM reserves the right to update the information shown here. For the latest information regarding levels of supported software, refer to:
http://www.software.ibm.com/ts/mqseries/platforms/supported.html

For the latest IBM statement regarding Year 2000 readiness, refer to:

http://www.ibm.com/ibm/year2000/

Chapter 2

MQSeries Integrator Overview

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

- MQSeries
- Formatter
- Rules
- 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. Additionally, MQSeries 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.

Formatter

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 NEONFormatter's database is done through the MQSeries Integrator 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.

Rules

Use NEONRules to manage 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 the 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, see 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, NEONFormatter, and NEONRules in a generic server process. The MQSeries Integrator Rules daemon processes messages from an MQSeries input queue, uses NEONFormatter to parse messages, uses NEONRules 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

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.

What is Formatter?

NEONFormatter has two main functions: parsing and reformatting.

- Parsing separates input messages into individual fields.
- Reformatting transforms input messages into an output message with a different format.

NEONFormatter 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 NEONFormatter GUI tool or the Formatter management API functions.

The NEONFormatter 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.

NEONFormatter 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 NEONFormatter. 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 1.0database only.

The NEONFormatter 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 that describes 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.

NEONFormatter supports several data types including ASCII String, ASCII Numeric, and Binary. See *Data Types* on page 259 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. For example, a literal can represent numbers or strings that provide an actual value instead of representing possible values. Literals might contain only 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, 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 comply with the POSIX 1003.2 standard for regular expressions.

For more information on literals and regular expressions, see 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 NEONFormatter 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 can 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. Use either the

NEONFormatter GUI or Formatter Management APIs to define and configure format descriptions.

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

Format Storage

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

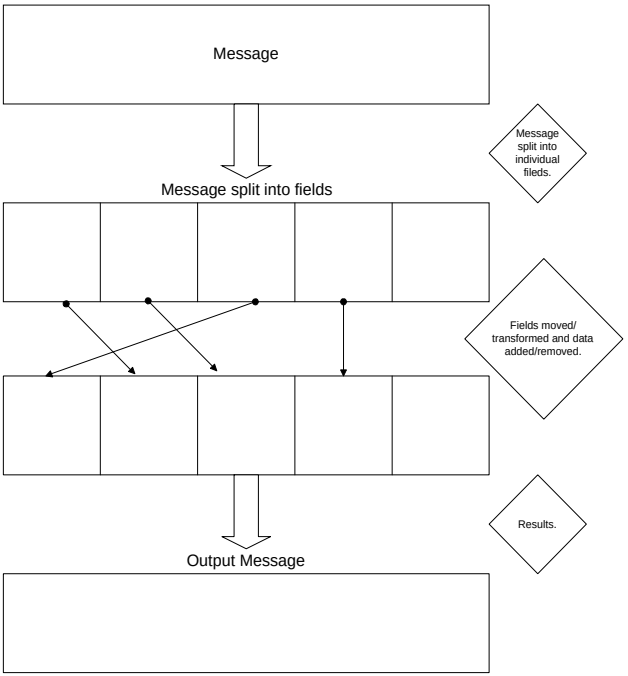
Possible transformations NEONFormatter can handle include:

- 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

NEONFormatter 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.

Alternatively, 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

MQSeries Integrator Version 1.1 was developed using shared libraries and DLLs; therefore, you do not have to recompile the binaries each time you update the software with a change. For example, when NEON sends code enhancements, the code is brought in at run time, so all you need to do is replace the library.

The sqlsvses.cfg file contains information used by Rules and NEONFormatter.The shared libraries and configuration files are explained in the following sections.

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 *Rules* on page 87.

Shared Libraries/DLLs

In Windows NT, libraries are called Dynamic Link Libraries (DLLs). UNIX refers to them as shared libraries (files with .so or .sl extensions), as does AIX (files with .a extensions). With these shared libraries, binaries are smaller and code changes are easier than using DLLs. Because the code is brought in at run time, the binaries do not have to be recompiled to incorporate a change. Another benefit of the shared libraries is the ability to create and add User Exits to NEONFormatter without relinking the executables.

To use the shared libraries, MQSeries Integrator Version 1.1 is required. The new libraries must be compatible with the current version of MQSeries

Integrator. For example, if library names or APIs are different, the shared libraries do not work.

Note:
This functionality is not compatible with earlier versions of MQSeries Integrator.

WARNING!
Do not move the libraries. The executables search for them in a specific directory or folder. If you move or delete the libraries, the executables are rendered useless.

Running NNWhich

NNWhich displays version information along with the path specified in an environment variable. This environment variable must include the directory or folder containing the MQSeries Integrator shared libraries.

Environment variables vary according to the platform, which are shown in the following list:

- Solaris: LD_LIBRARY_PATH
- AIX: LIBPATH
- HP-UX: SHLIB_PATH
- Windows NT: PATH

Syntax and Example Output (UNIX)

```
NNWhich librule.so
/usr/lib/Neonet/libnnrule.so(/usr/lib/Neonet/librule.so.1)
R4_0_56 1998/06/12 12:56
```

Note:
Because MQSeries Integrator does not provide pointers to the DLLs, only one path is output when you run NNWhich.

Replacing the Library

When you receive a code enhancement from Technical Support, follow these steps to replace the library and update the code:

UNIX

1. Verify that you have received the correct .so.x file (x indicates the version).
2. Switch to the directory that contains the shared libraries.
3. Run the SharedLinker that is sent with the .so.x file.

   This creates a .so file that points to the .so.x code enhancement file. The executable will now link to the new code.

Windows NT

1. Verify that you have received the correct DLL version.
2. Switch to the directory that contains the DLLs.
3. Replace the old version with the new version.

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 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 or DB2.

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. For Oracle and DB2, there must be a colon after the password even though the last parameter is not used.

Encrypting the sqlsvses.cfg File

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

Modifying the Location of the sqlsvses File

The default location of the sqlsvses file is the local directory in which 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, but the default configuration file can not be sqlsvses.crypt.

One copy of sqlsvses.cfg can be set so that all directories point to it, eliminating the need for the file in every directory.

For example, on NT:

```
SET 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 message is displayed.

Editing the sqlsvses.cfg File

To give MQSeries Integrator the database information it needs for configuration, 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>
```

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

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

For Oracle and DB2 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. DB2 servers use database names or aliases instead of server names.

The following is a sample text line in the sqlsvses.cfg file for an Oracle server:

```
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>.

Changing the sqlsvses.cfg File

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

Required Components

This utility requires the following:

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

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.

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

Importing and Exporting Formats

NNFie

NNFie is a command that exports format definitions from a database to an export file and imports from the export file into a database. The UNIX command for running the NNFie script is as follows:

NNFie.sh

Note:
To use NNFie, UNIX users must have write permissions to the current directory.

The NT command for running NNFie.exe is as follows:

NNFie.cmd

The export file for NNFie is not interchangeable with the files created by the GUI. NNFie can import data from a MQSeries Integrator Version 1.1 export file into a MQSeries Integrator 1.1 database. NNFie 1.1 exports data only from a MQSeries Integrator Version 1.1 database.

Note:
NNFie, NNRie, and sqlsvses.cfg must be in the same directory as NNFie.sh.

WARNING!
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.

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

Syntax

When entering NNFie related commands, maintain the order of options as they are listed below.

```
NNFie
 ((-C <command file name>)
 (-i <import file name> [-T] [ -o|-g|-n|-4]
 [-s <session name>])
 (-e <export file name> [-m <format name>+] [-q "comment"]
 [-Q <Comment file name>] [-w <number>] [-s <session name>])
```

```
 (-t <import file name> [-s <session name>])
 (-I <import file name> [-s <session name>]))
```

```
[ ] represents optional
() represents grouping
| represents XOR
```
+ represents one or more
<> means replace with user-provided data

Note:
The options to NNFie are positionally important. The following command is correct:

```
>nnfie -e myfile -m myformatname -s nnfie
```
The following command is positionally incorrect:

```
>nnfie -e my file -s nnfie -m myformatname
```

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. WARNING: Command line option -C puts import/ export command options in a text file. Do not use quotation marks around names (e.g., format name, session name, etc.) in the text file. Also, do not use back slashes in command lines.

Name	Mandatory/ Optional	Description
-i [<import file>]	Mandatory for Import	This parameter is required to import data from the named file and is mutually exclusive from -e. The named file default is NNFie.exp. If you use the command line option -i, then the following options are available to you: [-T] [-o | -g | -n | -4]. These additional options are described below the parameters table.
-e [<export file>]	Mandatory for Export	This parameter is required to export data from the named file and is mutually exclusive from -i. The named file default is NNFie.exp. If you use the command line option -e, then the following options are available to you: -q, -Q, -w, and -m. These additional options are described below the parameters table.
-s [<session name>]	Optional	Name of session in sqlsvses.cfg. Defaults to NNFie.
-I<import file name>	Mandatory	Writes description of all conflicts in import file to NNFie.log.
-t <import file name>	Mandatory	Writes an inventory of the import file to NNFie.log.

Import Options	Mandatory/ Optional	Description
-T	Optional	Loads import file as one transaction. If an import failure for one component is detected, then the entire import is rolled back. The default behavior is a transaction boundary for each component.

Import Options	Mandatory/ Optional	Description
-o	Optional	Overwrites all conflicts and replaces all components of same name with those in the export file.
-g	Optional	Ignores all conflicts and uses existing component definitions.
-n	Optional	Implements the interactive conflict resolution option. NNFie defaults to -n if no options are selected.
-4	Optional	Use R4_0 conflict resolution if a component in the export file conflicts with current data in the database. Do not import the new component but flag it in the error file and do not import any components that rely on the conflicting component.

Export Options	Mandatory/ Optional	Description
-q	Optional	Adds comments within quotes to top of the export file.
-Q	Optional	Adds contents of <comment file> to top of export file.
-w	Optional	Sets maximum line length in export file. Default value is 80.
-m [<message type>]	Optional	Specifies the message type to export. By default, exports all messages types within the specified application group.

Import Syntax

Case 1: Import a format

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

Note:
NNFie stores error messages in the NNFie.log file. If a component fails to import, the line containing an error from the export file is written to NNFie.err.

Export Syntax

Case 2: Export an entire database

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

Case 3: Export a single format

```
$ NNFie.sh -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:
NNFie.sh cannot be used to export formats from an earlier version to an newer version.

Conflict Resolution

A conflict occurs when an imported component does not match an existing component in the database of the same name and type.

Note:
NNFie is not designed to import or export databases that are corrupt or have unresolved issues with the data.

Inventory Components

When a component is overwritten, the component definition within the export file is imported into the database. To implement the batch overwrite conflict resolution, type the following syntax:

```
NNFie -i <filename> -o
```

When a component is either ignored or skipped, the component in export file **is not** imported into the database. To implement the batch Ignore/Skip conflict resolution, type the following syntax:

```
NNFie -i <filename> -g
```

Note:
Components that are skipped may cause the import of supporting components that will not be used. Overwriting existing components may cause existing components not to be used. This does not affect the integrity of the database, but requires use of a clean-up utility.

When the system identifies a conflict in interactive mode, it displays a description of both the existing and import components, and you are presented with three options: Overwrite, Ignore, and Rename the imported component. If you select the Rename option, all references to this component within the export file are updated.

To implement the interactive conflict resolution option, type the following syntax:

```
NNFie -i <filename> -n
```

To implement conflict resolution used in release 4.0, where no conflicting components are imported, type the following syntax:

```
NNFie -i <filename> -4
```

If you do not choose a conflict resolution option, the interactive resolution is used as the default. All conflicts and their resolution are reported to the NNFie.log file.

Example

```
Literal:"MyLiteral" conflicts with an existing Formatter
element!
```

```
literalLength ( existing=2 | incoming=3 )
Overwrite, Ignore or Rename component (OIR): R
Please enter new component name: MyLiteral_NewValue
```

Troubleshooting Export Failures

You can take inventory of the components contained in an NNFie export file. This inventory option produces a component inventory listing in the file named NNFie.log. The command syntax is as follows:

```
NNFie -t <filename>
```

You have the option of identifying all conflicts without importing any data. This test import function allows you to verify the contents of export files within working databases, thus facilitating easy validation for archiving. Any conflicts are recorded in the NNFie.log file. To use this conflict report, type the following syntax:

```
NNFie -I <filename> -k
```

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. Nonconflict errors.

Nonconflict Error Message (not component specific)

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

ERROR: <error message>

Nonconflict 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>
```

Checking Component Definitions

1. Run the NEONFormatter consistency checker.
2. Fix any problems in the current database.
3. Verify that is enough storage in memory for the new 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> )
```

Conditional Branching

When you use the Export by Name option during the export of formats, each output control that uses conditional branching exports the output controls associated with that output format, as defined by the rules entries.

NNFie File Layout

By removing NNFie encryption, you can access and interpret NNFie export files through a text editor. In past releases, the only out-of-the-box access to the Formatter database configurations have been through the NEONFormatter GUI. Now, with the export files in a readable form, you can write or modify scripts that create formatter components. Moreover, in prior releases, users were required to use the NEONFormatter GUI to respond to an import error

report. Because the import files generated from the current release of MQSeries Integrator are not encrypted, you can navigate and modify them.

WARNING!
Although we recommended using the NEONFormatter GUI instead of modifying the raw export files, this section provides the necessary information for an advanced user who is experienced with NNFie import/export formats to make changes.

Note:
Encryption has been removed from NNRie as well NNFie.

NNFie Header

The first five lines of the file, which begin with hash marks (#), are used to enter comments containing the following information:

1. GMT Time of creation
2. Version of NNFie
3. Database login data
4. Database server
5. OS data

Lines that begin with a number character (#) are interpreted as comments and are ignored during import. If you want to add comments or file concatenations to the header during export, type one of the following options at the command prompt:

NNFie -e <filename> -q "additional comment between quotes"

NNFie -e <filename> -Q <comment file>

Format of Export Data

Note:
Refer to the appropriate header files for the enumerated types.

A continuation marker breaks the component definition into several lines within the export file. If the last character on the line is a backslash (\), then the next line will be concatenated by the export file reader. The default line width is 80 characters, but you can specify an optional width by using the following command line option:

NNFie -e <filename> -w 120

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

Common Fields

There are three fields at the beginning of each formatter component definition.

The exclamation character **(!**) is used as a delimiter between the first three fields.

First Field

This field contains a NEONFormatter marker *F* and is used to indicate the beginning of a NEONFormatter component definition. *F* must appear at the beginning of every line, with the exception of comment lines, in the file.

Second Field

This field holds the release number of the defined component. The use of the version number to define components enables NNFie to support several revisions of export files.

Third Field:

The names of valid components for the integers in the third field are listed in the table below with the recommended releases (See Second Field regarding releases).

	Field Name	Release
1	Format	4.0
2	Input Control	4.0.1
3	Output Control – obsolete, use Output Master	3.x
4	Delimiter – obsolete, use Literal	3.x
5	Field	4.0
6	User Defined Type	4.0
7	Literal	4.0
8	Output Master	4.0
9	Default Control	4.0
10	User Exit Control	4.0
11	Fix Control	4.0
12	Length Control	4.0
13	Math Control	4.0
14	Substitution Control	4.0
15	Substring Control	4.0
16	Trim Control	4.0
17	Collection Control	4.0

Components of the Format Definition

Each NEONFormatter item export data takes up one line.

When a string is used as a field type it is typically 32 characters or less.

When an encoded hex is used as a field type it is typically up to 254 characters. The only valid characters in this field are **0x** followed by **[0-9A-F]**.

When an integer defines a code for an enumerated type, refer to fmtcodes.h in the include directory for valid entries. All definitions using enumerated type have the fixed type defined as **enum**.

NNFie uses the Management API to populate the database with Formatter components. Refer to the ***Programming Reference for NEONFormatter*** for explanation of field values. In almost all cases, NNFie uses the information in the export file to populate the NNFMgr<Component Type>Info structure.

Refer to the ***Programming Reference for NEONFormatter*** for issues not covered in this section.

In the component descriptions that follow, these conventions are used:

\: Continuation marker.

(...)+: Items within the parenthesis exist one or more times.

(...)*: Items within the parenthesis exist zero or more times.

Flat Input Format

F!<Version No — number>!1!\

Note:
The portion of the format below relates to the NNFMgrFormatInfo structure.

<Format Name — string>,\

1,\ // Input Indicator

0!\ // Compound Indicator

Note:
The portion of the format below relates to the NNFMgrFlatFormatInfo structure.

<Decomposition ID — int>,\

<Length ID — int>,\

<Termination ID — int>,\

<Delimiter Name — string>!\

<Number of Input Field / Control Pairs — integer>!\

(

Note:
The portion of the format below relates to the NNFMgrInFieldInfo structure.

<Format Name — string>,\

<Field Name — string>,\

<Control Name — string>!\

)+

Example

F!4.0!1!Flat_IF,1,0!1,0,0,NONE!2!Flat_IC,alpha,alpha_IC!Flat_IC,numeric,numeric_IC!

Flat Output Format

F!<Version No — number>!1!\

Note:
The portion of the format below relates to the NNFMgrFormatInfo structure.

<Format Name — string>,\

0,\ // Input Indicator

0!\ // Compound Indicator

Note:
The portion of the format below relates to the NNFMgrFlatFormatInfo structure.

<Decomposition ID —int>,\

<Length ID — int>,\

<Termination ID — int>,\

<Delimiter Name — string>!\

<Number of Output Field / Control Pairs — integer>!\

(

Note:
The portion of the format below relates to the NNFMgrOutFieldInfo structure.

<Format Name — string>,\

<Field Name — string>,\

<Control Name — string>,\

<Access Mode — int>,\

<Subscript — integer>,\

<Infield Name — string>!\

)+

Example

F!4.0!1!Flat_OF,0,0!1,0,0,NONE!3!Flat_OC,alpha,alpha_OC,1,0,alpha!Flat_OC,alpha,alpha_OC,1,0,alpha!Flat_OC,numeric,numeric_OC,4,0,numeric!

Compound Format

F!<Version No — number>!1!\

Note:
The portion of the format below relates to the NNFMgrFormatInfo structure.

<Format Name — string>,\

<Input Indicator ID —int>,\

1!\ / / Compound Indicator

<Number of Child Formats — integer>!\

(

Note:
The portion of the format below relates to the NNFMgrRepeatFormatInfo structure.

<Parent Format Name — string>,\

<Child Format Name — string>,\

<Optional Indicator ID — integer>,\

<Repeat Indicator ID — integer>,\

<Repeat Termination ID — integer>,\

<Repeat Delimiter Name — string>,\

<Repeat Field Name — string>!

)+

Example

F!4.0!1!CompRep_IF,1,1!1!CompRep_IF,Flat_IC,0,1,1,=,0,NONE!

Input Control (4.0.1)

F!4.0.1!2!\

Note:
The portion of the format below relates to the NNFMgrParseControlInfo structure.

<Control Name — string>,\

<Optional Indicator ID —int>,\

<Field Type ID — int>,\

<Data Type Name — string>,\

<Base Data Type ID — int>,\

<Custom Date Time Format— string>,\

<Data Termination ID — int>,\

<Data Delimiter Name — string>,\

<Data Length — number>,\

<Tag Type ID — int>,\

<Tag Termination ID — int>,\

<Tag Length — integer>,\

<Tag Literal Name — string>,\

<Tag Value — encoded hex>,\

<Tag Delimiter Name — string>,\

<Length Type ID —int>,\

<Length Termination ID —int>,\

<Length Length — integer>,\

<Length Delimiter Name — string>,\

<Decimal Location — integer>,\

<Year Cut Off — integer>,\

<Validation Parameter Name — string>!\

<Number of Name/Value Pairs — integer>!\

(

<Name — string>,\

<Value — string>!\

)+

Example

F!4.0.1!2!alpha_IC,0,2,String,0,,2,NONE,6,0,0,3,tag,0x544147,NONE,0,0,0,NONE,0,101,!0!

Field

F!<Version No — number>!5!\

Note:
The portion of the format below relates to the NNFMgrFieldtInfo structure.

<Field Name — string >,\

<Comment — string >!

Example

F!4.0!5!numeric,Numeric field!

User Defined Type

F!<Version No — number>!6!\

Note:
The portion of the format below relates to the NNFMgrUserDefTypeInfo structure.

<Type Name — string >,\

<Native Type — string >,\

<Validation Routine Name — string>!

Example

F!4.0!6!Sample_UserDefinedType,String,UserDefinedTypeValidation!

Literal

F!4.0!7!\

<Literal Name — string >,\

Note:
The portion of the format below relates to the NNFMgrLiteralInfo structure.

<Value - ASCII — encoded hex >,\

<Value Length — integer>!

Example

!F!4.0!7!tag,0x544147,3!

Output Master

F!4.0!8!\

Note:
The portion of the format below relates to the NNFMgrOutMstrCntlInfo structure.

<Master Name — string>,\

<Optional Indicator ID —int>,\

<Field Type ID — int>,\

<Data Type Name — string>,\

<Data Attribute ID —int>,\

<Base Data Type ID — int>,\

<Tag Type ID — int>,\

<Tag Literal Name — string>,\

<Tag Value — ASCII-encoded hex>,\

<Tag Value Length — integer>,\

<Tag-before-Length Indicator ID — int>,\

<Length Type ID — int>,\

<Operation Type ID — int>,\

<Field Comparison Literal Name — string>,\

<Field Comparison Value — ASCII-encoded hex>,\

<Field Comparison Value Length — integer>,\

<Child Control Name — string>,\

<Child Control Type ID —enum NNCntlType>!

Example

F!4.0!8!alpha_OC,1,1,String,,0,0,NONE,0x00,0,0,0,0,NONE,0x00,0,NONE,0!

Default Control

F!4.0!9!\

Note:
The portion of the format below relates to the NNFMgrDefaultCntlInfo structure.

<Control Name — string>,\

<Literal Name — string>,\

<Value — ASCII-encoded hex>,\

<Value Length — integer>!

Example

F!4.0!9!Sample_DefaultCntl,Literal,0x4C69746572616C,7!

Exit Control

F!4.0!10!\

Note:
The portion of the format below relates to the NNFMgrUserExitCntlInfo structure.

<Control Name — string>,\

<Exit Routine Name — string>!

Example

F!4.0!10!Sample_UserExitCntl,ExitRoutineName!

Fix Control

F!4.0!11!\

Note:
The portion of the format below relates to the NNFMgrPrePostFixCntlInfo structure.

<Control Name — string>,\

<Literal Name — string>,\

<Value — ASCII-encoded hex>,\

<Value Length — integer>,\

<Place ID — enum NNFPrePostFix>,

<NULL Action Indicator — int>!

Example

F!4.0!11!Sample_FixCntl,Space,0x20,1,1,0!

Length Control

F!4.0!12!\

Note:
The portion of the format below relates to the NNFMgrLengthCntlInfo structure.

<Control Name — string>,\

<Pad Literal Name — string>,\

<Pad Value — ASCII — encoded hex>,\

<Value Length — integer>!

Example

F!4.0!12!Sample_LengthCntl,12,Space,0x20,1!

Math Control

F!4.0!13!\

Note:
The portion of the format below relates to the NNFMgrMathExpCntlInfo structure.

<Control Name — string>,\

<Decimal Precision — integer>,\

<Rounding Mode ID —int>!\

<Math Segment Count — integer>!\

(<Expression — string>!)+

Example

F!4.0!13!Sample_MathCntl,2,0!1!Field_1 * Field_2!

Substitution Control

F!4.0!14!\

Note:
The portion of the format below relates to the NNFMgrSubstituteCntlInfo structure.

<Control Name — string>,\

<Input Literal Name — string>,\

<Input Value — ASCII — encoded hex>,\

<Input Value Length — integer>,\

<Output Literal Name — string>,\

<Output Value — ASCII — encoded hex>,\

<Output Value Length — integer>,\

<Output Value Type ID —int>!\

<Substitute Count — integer >! \

(
<Control Name — string>,\

<Input Literal Name — string>,\

<Input Value - ASCII-encoded hex>,\

<Input Value Length — integer>,\

<Output Literal Name — string>,\

<Output Value - ASCII-encoded hex>,\

<Output Value Length — integer>,\

<Output Value Type ID — int>!

)*

Example

F!4.0!14!Sample_SubstituteCntl,NONE,0x00,0,NONE,0x00,0,1!3!Sample_Substituteℂntl,Space,0x20,1,X,0x58,1,1!Sample_SubstituteCntl,-,0x2D,1,_,0x5F,1,1!

Substring Control

F!4.0!15!\

Note:
The portion of the format below relates to the NNFMgrSubstringCntlInfo structure.

<Control Name — string>,\

<Start — integer>,\

<Length — integer>,\

<Pad Literal Name — string>,\

<Pad Value — ASCII-encoded hex>,\

<Pad Value Length — integer>!

Example

F!4.0!15!Sample_SubstringCntl,5,6,NONE,0x00,0!

Trim Control

F!4.0!16!\

Note:
The portion of the format below relates to the NNFMgrTrimCntlInfo structure.

<Control Name — string>,\

<Trim Character Literal Name — string>,\

<Trim Character Value — ASCII-encoded hex>,\

<Trim Character Value Length — integer>,\

<Trim Location ID — enum NNFTrim>!

Example

F!4.0!16!Sample_TrimCntl,Space,0x20,1,2!

Collection Control

F!4.0!17!\

<Control Name — string>,\

<Collection Count — integer>!\

(

<Child Control Name — string>,\

<Child Control Type — enum NNTcntlType>!\

)+

Sample Data

F!4.0!5!alpha,!

F!4.0!5!numeric,!

F!4.0!7!=,0x3D0000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000,1!
F!4.0!8!alpha_OC,1,1,String,,0,0,NONE,0x0000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000,0,0,0,0,NONE,0x00000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000,0,NONE,0!

F!4.0!8!numeric_OC,1,1,String,,0,0,NONE,0x00000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000,0,0,0,0,NONE,0x000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000,0,NONE,0!

F!4.0.1!2!alpha_IC,0,1,String,0,,2,NONE,6,0,0,0,NONE,0x000000EFFFE860000
00000000A95DC00101E200000000000006B54C00000001EFFFF01400000101EFF
FF014EF3C71010000000010000000100101E20EFFFF0DCEFFFF0DCEFFFE85000
0B2E60EF07D6D800101E20EF3C717800000000EFFFE854000FA841EFFFF088E
FFFF088EFFFF01400000000EFFFF014EFFFF0DC0000000000,NONE,0,0,0,NO
NE,0,101,!0!

F!4.0.1!2!numeric_IC,0,1,Numeric,0,,2,NONE,8,0,0,0,NONE,0x000000EFFFE8
6000000000000A95DC00101E200000000000006B54C00000001EFFFF0140000010
1EFFFF014EF3C71010000000010000000100101E20EFFFF0DCEFFFF0DCEFFFE
850000B2E60EF07D6D800101E20EF3C717800000000EFFFE854000FA841EFFFF

F088EFFFF088EFFFF01400000000EFFFF014EFFFF0DC0000000000,NONE,0,0,0,NONE,0,101,!0!

F!4.0!1!Flat_IC,1,0!1,0,0,NONE!2!Flat_IC,alpha,alpha_IC!Flat_IC,numeric,numeric_IC!F!4.0!1!Flat_OC,0,0!1,0,0,NONE!3!Flat_OC,alpha,alpha_OC,1,0,alpha!Flat_OC,alpha,alpha_OC,1,0,alpha!Flat_OC,numeric,numeric_OC,4,0,numeric!F!4.0!1!CompRep_IF,1,1!1!CompRep_IF,Flat_IC,0,1,1,=,0,NONE!F!4.0!1!CompRep_OF,0,1!1!CompRep_OF,Flat_OC,0,1,1,=,0,NONE!

NNFie Error and Format Error Messages

NNFie Error Messages

Code	Error Name	Error Message	Error Explan-ation	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.
-4002	NNFIEE_NO_IMPORT_FILE	No import files by the given name exist	The specified import file does not exist.	Create the file or check the accuracy of the input file name.
-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 get/return 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.

Code	Error Name	Error Message	Error Explan-ation	Response
-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 importing into a populated format database, 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.

Code	Error Name	Error Message	Error Explan-ation	Response
-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.
-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.

Code	Error Name	Error Message	Error Explan-ation	Response
-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.

Code	Error Name	Error Message	Error Explan-ation	Response
-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.
-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.

Code	Error Name	Error Message	Error Explan-ation	Response
-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 Explan-ation	Response
-4201	NNFIEE_GetFormat	GetFormat failed	The flat or compound format was not accessible in the database through the Formatter Management NNFMgr GetFormat API.	Use the secondary Formatter Management API error code to resolve the problem.

Code	Error Name	Error Message	Error Explan-ation	Response
-4202	NNFIEE_GetFirst Format	GetFirst Format failed	The first flat or compound format was not accessible in the database through the NNFMgr GetFirst Format API.	Use the secondary Formatter Management API error code to resolve the problem.
-4203	NNFIEE_GETNEXT FORMAT	GetNext Format failed	The next flat or compound format was not accessible in the database through the NNFMgr GetNext Format API.	Use the secondary Formatter Management API error code to resolve the problem.
-4204	NNFIEE_GetFirstFieldF romInputFormat	GetFirstField FromInput Format failed	The first field associated with a flat input format was not accessible in the database through the NNFMgr GetFirstField FromInput Format API.	Use the secondary Formatter Management API error code to resolve the problem.

Code	Error Name	Error Message	Error Explan-ation	Response
-4205	NNFIEE_GetNextField FromInputFormat	GetNextField FromInput Format failed	The next field associated with a flat input format was not accessible in the database through the NNFMgr GetNextField FromInput Format API.	Use the secondary Formatter Management API error code to resolve the problem.
-4206	NNFIEE_GetFirstField FromOutputFormat	GetFirstField FromOutput Format failed	The first field associated with a flat output format was not accessible in the database through the NNFMgr GetFirstField FromOutput Format API.	Use the secondary Formatter Management API error code to resolve the problem.
-4207	NNFIEE_GetNextField FromOutputFormat	GetNextField FromOutput Format failed	The next field associated with a flat output format was not accessible in the database through the API NNFMgr GetNextField FromOutput Format API.	Use the secondary Formatter Management API error code to resolve the problem.

Code	Error Name	Error Message	Error Explan-ation	Response
-4208	NNFIEE_GetFirstChild Format	GetFirstChild Format failed	The first child format of a compound format was not accessible in the database through the Formatter Management NNFMgr GetFirstChild Format API.	Use the secondary Formatter Management API error code to resolve the problem.
-4209	NNFIEE_GetNextChild Format	GetNextChild Format failed	The next child format of a compound format was not accessible in the database through the Formatter Management API NNFMgr GetNextChild Format.	Use the secondary Formatter Management API error code to resolve the problem. See the Formatter Management API error codes.
-4210	NNFIEE_GetOutput Control	GetOutput Control failed	The specified output control was not accessible in the database through the Formatter Management API NNFMgr GetOutput Control.	Use the secondary Formatter Management API error code to resolve the problem. See the Formatter Management API error codes.

Code	Error Name	Error Message	Error Explan-ation	Response
-4211	NNFIEE_GetFirst OutputControl	GetFirst OutputControl failed	The first output control was not accessible in the database through the Formatter Management API NNFMgr GetFirst Output Control.	Use the secondary Formatter Management API error code to resolve the problem. See the Formatter Management API error codes.
-4212	NNFIEE_GetNext OutputControl	GetNext OutputControl failed	The next output control was not accessible in the database through the Formatter Management API NNFMgr GetNext Output Control.	Use the secondary Formatter Management API error code to resolve the problem. See the Formatter Management API error codes.
-4213	NNFIEE_GetParse Control	GetParse Control failed	The specified parse/input control was not accessible in the database through the Formatter Management API NNFMgr GetParse Control.	Use the secondary Formatter Management API error code to resolve the problem. See the Formatter Management API error codes.

Code	Error Name	Error Message	Error Explan-ation	Response
-4214	NNFIEE_GetFirstParse Control	GetFirstParse Control failed	The first parse/input control was not accessible in the database through the Formatter Management API NNFMgr GetFirstParse Control.	Use the secondary Formatter Management API error code to resolve the problem. See the Formatter Management API error codes.
-4215	NNFIEE_GetNextParse Control	GetNextParse Control failed	The next parse/input control was not accessible in the database through the Formatter Management API NNFMgr GetNextParse Control.	Use the secondary Formatter Management API error code to resolve the problem. See the Formatter Management API error codes.
-4216	NNFIEE_GetDelimiter	GetDelimiter failed	The specified delimiter was not accessible in the database through the Formatter Management API NNFMgr GetDelimiter.	Use the secondary Formatter Management API error code to resolve the problem. See the Formatter Management API error codes.

Code	Error Name	Error Message	Error Explan-ation	Response
-4217	NNFIEE_GetFirst Delimiter	GetFirst Delimiter failed	The first delimiter was not accessible in the database through the Formatter Management API NNFMgr GetFirst Delimiter.	Use the secondary Formatter Management API error code to resolve the problem. See the Formatter Management API error codes.
-4218	NNFIEE_GetNext Delimiter	GetNext Delimiter failed	The next delimiter was not accessible in the database through the Formatter Management API NNFMgr GetNext Delimiter.	Use the secondary Formatter 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 Formatter Management API NNFMgr GetField.	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 Explan-ation	Response
-4220	NNFIEE_GetFirstField	GetFirstField failed	The first field was not accessible in the database through the Formatter Management API NNFMgr GetFirst Field.	Use the secondary Formatter 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 Formatter Management API NNFMgr GetNext Field.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4222	NNFIEE_Append FormatToFormat	Append FormatTo Format failed	The attempt to append one flat or compound format into a compound format failed using the Formatter Management API NNFMgr Append FormatTo Format.	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 Explan-ation	Response
-4223	NNFIEE_AppendField ToInputFormat	AppendField ToInput Format failed	The attempt to append a field to a flat input format failed using the Formatter Management API NNFMgr Append FieldToInput Format.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4224	NNFIEE_AppendField ToOutputFormat	AppendField ToOutput Format failed	The attempt to append a field to a flat output format failed using the Formatter Management API NNFMgr AppendField ToOutput Format.	Use the secondary Formatter 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 Formatter Management API NNFMgr AppendMath Expression.	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 Explan-ation	Response
-4226	NNFIEE_Append LookupEntry	Append LookupEntry failed	The attempt to append a lookup detail entry to an existing lookup control failed using the Formatter Management API NNFMgr Append LookupEntry.	Use the secondary Formatter 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 Formatter Management API NNFMgr CreateFormat.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4228	NNFIEE_CreateParse Control	CreateParse Control failed	The attempt to create a new parse/input control failed using the Formatter Management API NNFMgr CreateParse Control.	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 Explan-ation	Response
-4229	NNFIEE_CreateOutput Control	CreateOutput Control failed	The attempt to create a new output control failed using the Formatter Management API NNFMgr CreateOutput Control.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4230	NNFIEE_Create Delimiter	Create Delimiter failed	The attempt to create a new delimiter failed using the Formatter Management APINNFMgr Create Delimiter.	Use the secondary Formatter 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 Formatter Management API NNFMgr CreateField.	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 Explan-ation	Response
-4232	NNFIEE_SERIOUS_ ERROR_POSSIBLY_ DB_RELATED	GetErrorNo returned serious error number	General database error encountered using the Formatter Management APIs.	See Formatter Management API error code - 2604.
-4233	NNFIEE_GetDataType Name	GetData Typename 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.
-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.

Code	Error Name	Error Message	Error Explan-ation	Response
-4235	NNFIEE_GetFirstUser DefinedType	GetFirstUser DefinedType failed	The first user-defined type was not accessible in the database through the Formatter Management API NNFMgr GetFirstUser DefinedType.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4236	NNFIEE_GetNextUser DefinedType	GetNextUser DefinedType failed	The next user-defined type was not accessible in the database through the Formatter Management API NNFMgr GetNextUser DefinedType.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4237	NNFIEE_CreateUser DefinedType	CreateUser Defined Type failed	The attempt to create a new user-defined type failed using the Formatter Management API NNFMgr CreateUser DefinedType.	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 Explan-ation	Response
-4238	NNFIEE_GetFirst Literal	GetFirstLiteral failed	The first literal was not accessible in the database through the Formatter Management API NNFMgr GetFirst Literal.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4239	NNFIEE_GetNext Literal	GetNextLiteral failed	The next literal was not accessible in the database through the Formatter Management API NNFMgr GetNext Literal.	Use the secondary Formatter 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 Formatter Management API NNFMgr GetLiteral.	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 Explan-ation	Response
-4241	NNFIEE_GetFirstOut MstrCntl	GetFirstOut MstrCntl failed	The first output master control was not accessible in the database through the Formatter Management API NNFMgr GetFirstOut MstrCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4242	NNFIEE_GetFirst DefaultCntl	GetFirst DefaultCntl failed	The first default control was not accessible in the database through the Formatter Management API NNFMgr GetFirst DefaultCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4243	NNFIEE_GetFirstUser ExitCntl	GetFirstUser ExitCntl failed	The first user exit control was not accessible in the database through the Formatter Management API NNFMgr GetFirst UserExitCntl.	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 Explan-ation	Response
-4244	NNFIEE_GetFirstPre PostFixCntl	GetFirstPre PostFix Cntl failed	The first prefix / postfix control was not accessible in the database through the Formatter Management API NNFMgr GetFirstPre PostFixCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4245	NNFIEE_GetFirst SegmentFromMathExp Cntl	GetFirst SegmentFrom MathExpCntl failed	The first segment of the math expression detail control was not accessible in the database through the Formatter Management API NNFMgr GetFirst SegmentFrom 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 Explan-ation	Response
-4246	NNFIEE_Append SegmentToMathExp Cntl	Append SegmentTo MathExpCntl failed	The attempt to append a math expression detail entry to an existing math expression failed using the Formatter Management API NNFMgr Append SegmentMath ExpCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4247	NNFIEE_GetFirst SubstituteCntl	GetFirst SubstituteCntl failed	The first substitute control was not accessible in the database through the Formatter Management API NNFMgr GetFirst SubstituteCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4248	NNFIEE_GetFirstSub StringCntl	GetFirstSub StringCntl failed	The first substring control was not accessible in the database through the Formatter Management API NNFMgr GetFirstSub StringCntl.	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 Explan-ation	Response
-4249	NNFIEE_GetFirstTrim Cntl	GetFirstTrim Cntl failed	The first trim control was not accessible in the database through the Formatter Management API NNFMgr GetFirstTrim Cntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4250	NNFIEE_GetFirst CollectionCntl	GetFirst CollectionCntl failed	The first output collection control was not accessible in the database through the Formatter Management API NNFMgr GetFirst CollectionCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4251	NNFIEE_AppendCntl ToCollectionCntl	AppendCntl ToCollection Cntl failed	The attempt to append an output operation to an output operation control failed using the Formatter Management API NNFMgr AppendCntl ToCollection Cntl.	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 Explan-ation	Response
-4252	NNFIEE_GetFirstCntl FromCollection	GetFirstCntl From Collection failed	The first output operation collection control was not accessible in the database through the Formatter Management API NNFMgr GetFirstCntl From Collection.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4253	NNFIEE_GetFirst LengthCntl	GetFirst LengthCntl failed	The first length control was not accessible in the database through the Formatter Management API NNFMgr GetFirst LengthCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4254	NNFIEE_GetFirstMath ExpCntl	GetFirstMath ExpCntl failed	The first math expression control was not accessible in the database through the Formatter Management API NNFMgr GetFirstMath ExpCntl.	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 Explan-ation	Response
-4255	NNFIEE_GetNextOut MstrCntl	GetNextOut MstrCntl failed	The next output master control was not accessible in the database through the Formatter Management API NNFMgr GetNextOut MstrCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4256	NNFIEE_GetOutMstr Cntl	GetOutMstr Cntl failed	The specified output master control was not accessible in the database through the Formatter Management API NNFMgr GetOutMstr Cntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4257	NNFIEE_GetNext DefaultCntl	GetNext DefaultCntl failed	The next default control was not accessible in the database through the Formatter Management API NNFMgr GetNext DefaultCntl.	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 Explan-ation	Response
-4258	NNFIEE_GetDefault Cntl	GetDefault Cntl failed	The specified default control was not accessible in the database through the Formatter Management API NNFMgr GetDefault Cntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4259	NNFIEE_GetNextUser ExitCntl	GetNextUser ExitCntl failed	The next user exit control was not accessible in the database through the Formatter Management API NNFMgr GetNextUser ExitCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4260	NNFIEE_GetUserExit Cntl	GetUserExit Cntl failed	The specified user exit control was not accessible in the database through the Formatter Management API NNFMgr GetUserExit Cntl.	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 Explan-ation	Response
-4261	NNFIEE_GetNextPre PostFixCntl	GetNextPre PostFixCntl failed	The next prefix / postfix control was not accessible in the database through the Formatter Management API NNFMgr GetNextPre PostFixCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4262	NNFIEE_GetPrePostFix Cntl	GetPrePostFix Cntl failed	The specified prefix / postfix control was not accessible in the database through the Formatter Management API NNFMgr GetPrePostFix Cntl.	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 Explan-ation	Response
-4263	NNFIEE_GetNext SegmentFromMathExp Cntl	GetNext SegmentFrom MathExpCntl failed	The next segment of the math expression detail controls was not accessible in the database through the Formatter Management API NNFMgr GetNext SegmentFrom MathExpCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4264	NNFIEE_GetNext SubstituteCntl	GetNext SubstituteCntl failed	The next substitute control was not accessible in the database through the Formatter Management API NNFMgr GetSubstitute Cntl.	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 Explan-ation	Response
-4265	NNFIEE_GetSubstitute Cntl	GetSubstitute Cntl failed	The specified substitute control was not accessible in the database through the Formatter Management API NNFMgr GetSubstitute Cntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4266	NNFIEE_GetNext SubStringCntl	GetNextSub StringCntl failed	The next substring control was not accessible in the database through the Formatter Management API NNFMgr GetNextSub StringCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4267	NNFIEE_GetSubString Cntl	GetSubString Cntl failed	The specified substring control was not accessible in the database through the Formatter Management API NNFMgr GetSubString Cntl.	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 Explan-ation	Response
-4268	NNFIEE_GetNextTrim Cntl	GetNextTrim Cntl failed	The next trim control was not accessible in the database through the Formatter Management API NNFMgr GetNextTrim Cntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4269	NNFIEE_GetTrimCntl	GetTrimCntl failed	The specified trim control was not accessible in the database through the Formatter Management API NNFMgr GetTrimCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4270	NNFIEE_GetNextCntl FromCollection	GetNextCntl From Collection failed	The next output operation collection control was not accessible in the database through the Formatter Management API NNFMgr GetCntlFrom Collection.	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 Explan-ation	Response
-4271	NNFIEE_GetNext LengthCntl	GetNext LengthCntl failed	The next length control was not accessible in the database through the Formatter Management API NNFMgr GetNext LengthCntl.	Use the secondary Formatter 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 Formatter Management API NNFMgr GetLength Cntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4273	NNFIEE_GetNextMath ExpCntl	GetNextMath Exp Cntl failed	The next math expression control was not accessible in the database through the Formatter Management API NNFMgr GetNextMath ExpCntl.	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 Explan-ation	Response
-4274	NNFIEE_GetMathExp Cntl	GetMathExp Cntl failed	The specified math expression control was not accessible in the database through the Formatter Management API NNFMgr GetMathExp Cntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4275	NNFIEE_GetNext CollectionCntl	GetNext CollectionCntl failed	The next output collection control was not accessible in the database through the Formatter Management API NNFMgr GetNext CollectionCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4276	NNFIEE_GetCollection Cntl	GetCollection Cntl failed	The specified output collection control was not accessible in the database through the Formatter Management API NNFMgr GetCollection Cntl.	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 Explan-ation	Response
-4277	NNFIEE_GetUser DefinedType	GetUser DefinedType failed	The specified user-defined type was not accessible in the database through the Formatter Management API NNFMgr GetUser DefinedType.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4278	NNFIEE_GetNextMath Expression	GetNextMath Expression failed	The next math expression was not accessible in the database through the Formatter Management API NNFMgr GetNextMath Expression.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4279	NNFIEE_GetNextLook upEntry	GetNext LookupEntry failed	The next lookup entry control was not accessible in the database through the Formatter Management API NNFMgr GetNext LookupEntry.	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 Explan-ation	Response
-4280	NNFIEE_GetNextEntry FromSubstituteCntl	GetNextEntry From SubstituteCntl failed	The next substitute field segment from the substitute control was not accessible in the database through the Formatter Management API NNFMgr GetNext EntryFrom SubstituteCntl.	Use the secondary Formatter Management API error code to resolve the problem. See the Format Management API error codes.
-4281	NNFIEE_AppendEntry ToSubstituteCntl	AppendEntry ToSubstitute Cntl failed	The attempt to create a substitute field segment for the substitute control failed using the Formatter Management API NNFMgr Append EntryToSubsti tuteCntl.	Use the secondary Formatter 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

The following testing executables are provided with NEONFormatter:

- apitest
- 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 NEONFormatter interpreted a message.

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

For more information on msgtest and apitest, refer to ***MQSeries Integrator Programming Reference for NEONFormatter***.

The apitest Executable

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

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 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 UNIX file.

```
$ 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 thetranslated 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

NEONRules is dependent on NEONFormatter to parse messages for evaluation and has the following main functions:

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

NEONRules 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.

NEONRules is packaged as a library of C++ objects that have public functions, which 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.

Rule definitions describe how to parse a message using the format parameters (specified in NEONFormatter) 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 either the Rules GUI or Rules Management API functions.

The NEONRules graphical user interface (GUI) tool is a program used 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 primary 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 NEONRules 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 ensure data integrity.

The NNRie tool delivered with Rules is a command line tool that can be used to export rule definitions from a database to a file, and to import the exported file into a database. NNRie can import from a NEONetMQ Integrator 3.X and 4.X export file into an NEONetMQSeries Integrator 1.14.1.x database. NNRie Release 4.1.x exports data only from a 4.1.x database.

Remove encryption from the NNRie files to access and interpret these files without using the NEONRules GUI and customer support.

Application Groups

Application groups are logical divisions of rule sets for different business needs. You can define unlimited application groups. For example, you might want the 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 a rule is created, each rule is assigned a rule name and associated with an application group and message type. Each rule is uniquely identified by its application group / message type / rule name.

The following three items must be defined for each rule:

- Evaluation criteria — An expression that contains arguments and operators
- Subscription information — Subscriptions, actions, and options
- Permission information

Expressions, Arguments, Boolean, and Rules Operators

An expression (evaluation criteria) contains a list of fields, associated operators, and associated comparison data (either static values or other fields) connected with Boolean operators. An argument contains 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 NEONFormatter. 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 *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, you can retrieve associated subscriptions to be taken by the application. These subscriptions are the actions or commands, and the associated parameters are the 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 an action name. The action does not need to be unique, since all actions are intended to be executed in sequence. A single subscription can be shared by multiple 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.

Rule and Subscription Permissions

Rule and Subscription permissions restrict user access to individual complete rules or subscriptions or their components in the NEONRules 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 NEONRules 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

The two types of Rules APIs are:

- Rules APIs — Evaluates rules and retrieves 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.
- Rules Management APIs — Maintains 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

NEONet Release 4.1.x was developed using shared libraries and DLLs. Therefore, you do not have to recompile the binaries each time you update the software with a change (for example, when NEON sends code enhancements). The code is brought in at run time, so all you do is replace the library.

The sqlsvses.cfg file contains information used by NEONRules and NEONFormatter. The shared libraries and configuration files are explained in the following sections.

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.

DLLs/Shared Libraries

Windows NT calls the libraries Dynamic Link Libraries (DLLs), Unix refers to them as shared libraries (files with .so or .sl extensions), as does AIX (files with .a extensions). With these shared libraries, binaries are smaller and code changes are easier. Because the code is brought in at run time, the binaries do not have to be recompiled to incorporate a change. Another benefit of the shared libraries is the ability to create and add User Exits to Formatter without relinking the executables.

To use the shared libraries, you must have NEONet Release 4.1.x installed. The new libraries must be compatible with the current version of NEONet.

For example, if library names or APIs are different, the shared libraries do not work.

Note:
This functionality is not compatible with earlier versions of NEONet

WARNING!
Do not move the libraries. The executables search for them in a specific directory or folder. If you move or delete the libraries, the executables are rendered useless.

Running NNWhich

NNWhich is an executable that can help you determine the version of your libraries. When you run NNWhich, the version information is displayed, along with the path, specified in an environment variable. This environment variable must include the directory or folder containing the NEONet shared libraries. Environment variables vary according to platform:

- Solaris: LD_LIBRARY_PATH
- AIX: LIBPATH
- HP-UX: SHLIB_PATH
- Windows NT: PATH

Syntax and Example Output (Unix)

```
NNWhich librule.so
/usr/lib/Neonet/libnnrule.so(/usr/lib/Neonet/librule.so.1)
R4_0_56 1998/06/12 12:56
```

Note:
Because NEONet does not provide pointers to the DLLs, only one path is output when you run NNWhich on Windows NT.

Replacing the Library

When you receive a code enhancement from NEON Technical Support, complete the following steps to replace the library and update the code:

Unix

1. Verify that you have received the correct .so.x file (x indicates the version).
2. Switch to the directory that contains the shared libraries.
3. Run the shared linker script that accompanies the .so.x file.

   ```
   SharedLinker .so.x
   ```

   This creates a .so file that points to the .so.x code enhancement file. The executable will now link to the new code.

Windows NT

1. Verify that you have received the correct DLL version.
2. Switch to the directory that contains the DLLs.
3. Replace the old version with the new version.

sqlsvses.cfg File

The sqlsvses.cfg file is the default configuration file that contains information about the database and database server used for NEONetMQSeries 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 NEONetMQSeries Integrator components.

sqlsvses.cfg Parameters

Parameter	Description
session name	Database session name to be used by NEONetMQSeries Integrator executables or daemons. This can be any string as long as it is unique within the file.
server name	Server where the NEONetMQSeries Integrator database is resident.
user name (user id)	Database user name.
password	Database password.
database name	Database name where the NEONetMQSeries Integrator tables are resident (if applicable). This is not used for Oracle or DB2.

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 (:). For Oracle and DB2, there must be a colon after the password even though the last parameter is not used.

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 sqlsvses File Location

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, on NT:

```
SET 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 provide the messaging software with 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 and DB2 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. DB2 servers use database names or aliases 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 sqlsvses.cfg File Changes

To implement the changes made to the sqlsvses.cfg file, restart any applications using NEONetMQSeries 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 NEONetMQSeries Integrator users: NEONET_USER.

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

Creating Users

After you install NEONetMQSeries Integrator, you must create user names or assign NEONetMQSeries 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 NEONetMQSeries 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 using the sp_addlogin system procedure. Syntax for sp_addlogin is as follows:

```
sp_login 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, the group **NEONetUser** is created for NEONetMQSeries

Integrator users. To access NEONetMQSeries Integrator databases, users must be linked to the NEONetUser 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 is added. userName is the database user name.

Rule and Subscription Permissions

Note:
You must first create users before you grant permissions. For more information on creating users, refer to the *System Enhancements* section on page 97.

NNDBARuleOwnership

Permissions for Rules and Subscriptions should be managed through either the NEONRules GUI or Rules Management APIs. However, a tool is provided for System Administration. The NNRDBARuleOwnership utility allows the NEONetMQSeries 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 can connect to the NEONRules database.

Operations

To use the utility, type 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, type 1 at the prompt (shown as >). The utility displays:

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

If you select **1 List Rules Owned by a Certain User**, the utility lists the application group name, message type name and rule name of all rules owned by the specified user.

If you select **2 Change All Rules owned by User A to be Owned by User B**, the utility does not display this rule information.

To change rule ownership, type 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, type 3 at the prompt.

The utility displays:

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

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

To change subscription ownership, type 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 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 rules and subscriptions, refer to the ***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

Migrating Rules

WARNING!
This section explains how to migrate the rules data from a NEONet Release 3.1MQSeries Integrator 1.0 database to a MQSeries Integrator 1.1NEONet Release 4.1.x database. It is critical that you migrate formats before migrating rules; Rules uses format data for rule definitions. For information about format migration. See ***Migrating Formats***.

WARNING!
If you are using a database where case sensitivity is irrelevant, you cannot name components the same with only a change in case to identify them. For example, you cannot name one rule r1 and another rule R1. In this type of environment, you must make each item unique using something other than case differences.

If importing components into a case-insensitive database that were exported from a case-sensitive database, these differences will cause NNRie to fail during import because a conflict arises when two components are named the same with only case differences.

Overview

Use the NEONRules Import/Export Utility, NNRie, to export existing rules from a MQSeries IntegratorNEONet database and import them to a MQSeries IntegratorNEONet releaseVersion 14.1 database.

WARNING!
You must use the NNRie release that matches the release of the database from which you are exporting rules. For example, use NNRie release 3.x to export formats from a Rules 3.x database. NNRie release 4.1. can export only from a 4.1 database. Your NEONet 4.1.1 product CD-ROM contains NNRIE releases 3.1, 4.0, and 4.1.

Using NNRie, you can:

- Export a single rule identified by its corresponding application name, message type, and rule name.
- Export a single subscription identified by its corresponding application name, message type, and subscription name.
- Export entire rule sets, rules, and subscriptions identified by corresponding application group and message type names.
- Export all message types and their rule sets identified by the message type's application group name.
- Export all application groups and their associated message types and rules.
- Import a file exported by any 3.x or 4.x release of the NNRie program.

This program creates an export file that can be interchanged between platforms. All application groups and their associated message types and rules should be exported. The exported file can then be imported to the 14.1.1 database using NNRie releaseVersion 14.1.1.

NNRie release 4.1.1 can import a file created by exporting from a NEONet Rules database 3.0, 3.1, 3.2, 4.0, 4.1, and 4.1.1. NNRie skips inactive rule arguments when importing from a pre-4.1 NEONet database. NNRie release 4.1.1 cannot export from a database created before release 4.1.1. You must use NNRie from release 3.x to export from a 3.x database. If subscription Sub1 for Rule1 is different than Sub1 for Rule2, you are prompted to rename the second Sub1.

Preparation

Environmental Dependencies

This utility requires the following:

1. A supported RDBMS system previously installed.
2. The Rules database schema to have been previously created.
3. The Formatter database schema to have been previously created.

Export

Export requires the following:

1. Rules data in the database created via the Rules GUI or the Rules Management APIs.
2. Enough disk space to hold the output file. This file can be re-directed anywhere the system supports.

Import

Import requires the following:

1. Target database (MQSeries Integrator 1.1NEONet Release 4.1.x database) has been created and is large enough for the imported data.
2. The formats have been created via the NEONFormatter GUI, Formatter Management APIs, or NNFie.

Migration Process

The following steps are required to migrate your Rules from a 1.03.x or 4.0 database to a 1.14.1.x database.

1. Migrate your formats.
2. Set up your environment.
3. Check the consistency of your 1.03.x or 4.0 database using the consistency checker version of the database from which you are exporting.
4. Export your data from the 1.03.x or 4.0 database.
5. Modify your environment for import.
6. Import your formats into the 1.14.1.x database.
7. Check the consistency of your 1.14.1.x database using the consistency checker releaseversion 1.14.1.x.

Note:
The following section provides specific details about NNRie, the Rules import/export utility. The steps required to complete a full migration of

formats from a 1.03.x or 4.0 database to a 4.1.x database are discussed in detail in the *Installing NEONet MQSeries Integrator Installation and Configuration* guide.

Importing and Exporting Rules

NNRie

NNRie is a command line tool that you can use to export rule definitions and orphan subscriptions, which are subscriptions that are not associated with a rule, from a database to a file and to import the exported file into a database.

NNRie can import a NEONet 3.x and 4.x export file into a NEONet 4.1.x database. NNRie version 4.1.x exports data only from a NEONet 4.1.x database.

Note:
To use NNRie, UNIX users must have write permissions to the current directory.

Syntax

```
NNRie  ((-C [<command file name>] |
       -V |
       (-i <import file name>|-e <export file name>
       [[[-a <appname> [...]] [-m <msgname>] [...]] [-r
     <rulename>] [...]] [-S <subsname>] [...]]
       [-T [<trace file name>] ]
       [-l [<conflict report file name>] ]
       [-t [<inventory report file name>] ]
       [-f [<failure file name>] ]
       [-s <session name>]
       [-o]
       [-c <database configuration file name>])))
```

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. The 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.
-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 for a 3.x to 4.x1.0 to 1.1database. 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 NEONetMQSeries Integrator configuration file (See the -c option below). The default session tag is “nnrmie”.

Name	Mandatory/ Optional	Description
-o (overwrite flag)	Optional	The default behavior is off (do not overwrite). If this parameter is present during export, it overwrites 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 overwritten with the new definition if you have update permission. If you do not have update permission, 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.
-c <config file>	Optional	Indicates the name of the NEONetMQSeries 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.

Name	Mandatory/ Optional	Description
-S <subscription name>	Optional	Specifies the name of the subscription to export. This parameter also requires the -e, -a, and -m parameters to be set. This parameter can be repeated as many times as necessary to export multiple subscriptions.
-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.
-t [<inventory filename>]	Optional	Creates an inventory of an export file in NNRie.log (does no processing).
-T [<trace file name>]	Optional	Specifies the name of the trace file. Default trace file is NNRieT.log.
-O	Optional	Completely overwrites imported message types (import only). The default behavior is off (do not overwrite).
-l [<conflict report filename>]	Optional	Reports on any import conflicts. The default behavior is off (does no processing). Default file is NNRie.log.
-g	Optional	Ignore and do not import any conflicting rules and subscriptions.
-n	Optional	Implement interactive conflict resolution. The default behavior is on. MVS default is off.
-q <comments in double quotes>	Optional	Includes comments in an export file.

Name	Mandatory/ Optional	Description
-Q <comments file name>	Optional	Includes a file of comments in an export file. No default.
-f [<failure file>]	Optional	Specifies the failure file that contains lines not imported. The default file is NNRie.err.

Note:
If there are no -a, -m, -r, or -S options, the entire database exports.

Import Syntax

Case 1: Import a Rule

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

```
If the file fails to import, an error message is generated and
NNRie errors out.
```

Export Syntax

Export functionality is based on the type of parameters that are passed into the NNRie program. Each case listed below describes different ways to use the parameters to export information.

Case 2: Export an entire database

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

Case 3: Export a single application group

```
 $ NNRie -e [-a <app group name>]
```

The application group name exports and then each message type within the application group exports. The message type export includes all subscriptions

and rules in the specific application group / message type. This procedure is followed for each application group if multiple application group names are given.

Case 4: Export a message type for an application group

```
$ NNRie -e [-a <app group name>][-m <msgtype name>]
```

The application group name and message type name exports, then the rules export with the links to subscriptions. All subscriptions in the application group / message type export, whether they are linked to rules or not. If multiple message type names are given, the subscriptions and rules for each message type export.

Case 5: Export a single rule

```
$ NNRie -e [-a <app group name>] [-m <msgtype name>] [-r <rule
name>]
```

The rule's application group name and message type name exports. All subscriptions linked to the rule export with permissions, actions, and options and then the rule information exports with permissions, expressions, and links to subscriptions. If multiple rule names are given, the subscriptions linked to each rule export with no duplicates, and then the rules export.

Case 6: Export more than one rule

```
$ NNRie -e [-a <app group name>][-m <msgtype name>][-r <rule
name> <rule name>...]
```

Case 7: Export a single subscription

```
$ NNRie -e [-a <app group name>][-m <msgtype name>][-S
<subscription name>]
```

No rule information exports. The application group and message type name information exports and then the subscription information exports without the rule name. If multiple subscriptions are given, each subscription exports.

Remarks

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

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 creates or reads the file relative to the directory the user is in when the program was executed.

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.

WARNING!
If subscription conflicts exist, NNRie goes into interactive mode. Do not leave NNRie running unattended, unless you specify to overwrite existing rules and subscriptions with -o or message types with -O.

Summary of New Command Line Functions

All of the new functions below are optional.

To overwrite component by component, enter the following syntax:

```
NNRie -i <filename> -o
```

To run the batch Ignore / Skip conflict resolution, enter the following:

```
NNRie -i <filename> -g
```

To run the interactive conflict resolution option, enter the following:

```
NNRie -i <filename> -n
```

To run the “check only”, conflict reporting only option, enter the following:

```
NNRie -i <filename> -l (Writes to NNRie.log)
NNRie -i <filename> -l MyCLog.txt (Writes to MyCLog.txt)
```

To import and totally overwrite the application group message type pair in the database, enter the following syntax:

```
NNRie –i NNRie.exp –O
```

To trace the command that is about to be executed and save to a log file, enter the following:

```
NNRie –i NNRie.exp –T (Writes to NNRieT.log)
NNRie –i NNRie.exp –T trace.log (Writes to trace.log)
```

To produce an inventory of an export file, enter the following:

```
NNRie –t NNRie.exp (Writes to NNRie.log)
NNRie –i NNRie.exp –t inv.log (Writes to inv.log)
```

To add comments to the header of the Export file, enter the following:

```
NNRie -e <filename> -q "additional comment between quotes"
```

To add a file of comments to the header of the Export file, enter the following:

```
NNRie -e <filename> -Q <comment file>
```

Conflict Resolution

The added conflict resolution functionality allows you more flexibility when importing to a database already containing data. You can overwrite all or no rules and subscriptions as with past release. The new functionality allows you to go into an interactive mode, where you can choose to rename or overwrite heterogeneous rules/subscriptions. You can change the overwrite mechanism to overwrite the rule/ subscription as long as you have update permission. Ownership will no longer be required.

To implement the batch Overwrite conflict resolution, enter:

```
NNRie -i <filename> -o
```

To implement the batch Ignore/Skip conflict resolution, the syntax is:

```
NNRie -i <filename> -g
```

To implement the interactive conflict resolution option, the syntax is:

```
NNRie -i <filename> -n
```

To implement the conflict report option, the syntax is:

```
NNRie -i <filename> -l <optional filename>
```

If no conflict resolution option is chosen, the interactive resolution is used as the default.

The user should be able to replace an entire application group / message type pair by entering the following command:

Example: **NNRie –i NNRie.exp –O**

This command deletes each message type from the database that it encounters in the import file and all the Rules and subscriptions under it before importing new information. If it fails to delete because of rights violations or other problems, it returns an error message and does not import the new information.

Note:
If NNRie is not designed to import or export databases that are corrupt or have unresolved issues with the data.

Troubleshooting Importation Problems

If NNRie is unable to import an application group, message type, rule or subscription, the corresponding import information is written to the NNRie.err file. The NNRie.err file can be modified to fix the problem with the component and then used for reimportation. The reason for the component's failure to import is written to the NNRie.log file.

Refer to the documentation on the import file format for instructions on editing the NNRie.err file if it is version 1.14.1x. If the export file for

importation was released prior to version 4.1x, contact technical support to resolve the import problem.

Note:
The information in NNRie.err is not guaranteed to resolve your importation problem; rather, it should be viewed as a resource that will help you determine where the problem is in your import file.

NNRie.log

All conflicts and resolutions are reported to the NNRie.log file.

Conflict with Subscription: 'S3'

App Name: 'MsgTest'

Msg Name: 'MsgTest'

Subs in import file:

Owner: 'Public'

Comment: 'New Checking'

Subs in Database:

Owner: 'PUBLIC'

Comment: ''

Conflict Exists in : Comment

Optional Trace

This log file contains a progress report of the import. The Trace.log helps to pinpoint import problems.

Example: **`NNRie –i NNRie.exp –T trace.log`**

This command shows, line by line, what will be imported. If a process fails, the log stops within the errant process.

Trace Letters

The following letters define import and export components:

- A = Application group
- M = Message Type
- R = Rule
- n = permission (either rule or subscription)
- S = subscription - written to file or added to database
- C = action - written to file or added to database
- P = option - written to file or added to database
- s = subscription - read from file
- c = action - read from file
- p = option - read from file
- l =Subscription linked to a rule in the database

Inventory Export File

The inventory export file provides a tool to determine the items contained within an export file. The default log file is NNRie.log.

```
Example NNRie –t NNRie.exp (Writes to NNRie.log)
Example NNRie –i NNRie.exp –t MyInv.log (Writes to MyInv.log)
```

Inventory Report of NNRie Export File (nnrie.exp)

App Group: App1 Msg Type: AccDataIn Eval Type: NEONET_FORMATTER

Sub: SendFeeQ Comment:

Action: reformat

Option Name: INPUT_FORMAT Value: AccDataIn

Option Name: TARGET_FORMAT Value: AccDataOut

Action: putqueue

Option Name: OPT_TARGET_QUEUE Value: FeeQ

Option Name: OPT_MSG_TYPE Value: AccDataOut

Owner: gfullerton

Sub: SendPromoQ Comment:

Action: reformat

Option Name: INPUT_FORMAT Value: AccDataIn

Option Name: TARGET_FORMAT Value: AccDataOut

Action: putqueue

Option Name: OPT_TARGET_QUEUE Value: PromoQ

Option Name: OPT_MSG_TYPE Value: AccDataOut

Owner: gfullerton

Rule: MinBalCheck

Owner: gfullerton

Expr: (AccOpenDate DATETIME>= 19970601120000 | AccType STRING= FEE) & Balance INT< 200

Rule/Sub Link: Rule: MinBalCheck Sub: SendFeeQ

Rule: NoMinCheck

Owner: gfullerton

Expr: AccType STRING= FREE & AccOpenDate DATETIME< 19970601120000 & Balance INT>= 200

Rule/Sub Link: Rule: NoMinCheck Sub: SendFeeQ

Rule: CrazyRule

Owner: gfullerton

Expr: AccType EXIST

Rule/Sub Link: Rule: CrazyRule Sub: SendFeeQ

Rule: RealCrazyRule

Owner: gfullerton

Expr: AccOpenDate EXIST

Rule / Sub Link: Rule: RealCrazyRule Sub: SendFeeQ

Rule / Sub Link: Rule: RealCrazyRule Sub: SendPromoQ

App Group: App1 Msg Type: AccDataIn2 Eval Type: NEONET_FORMATTER

Rule: Rule1

Owner: gfullerton

Expr: AccType EXIST

App Group: MsgTest Msg Type: MsgTest Eval Type: NEONET_FORMATTER

Sub: AS1 Comment: "None"

Action: reformat

Option Name: INPUT_FORMAT Value: MsgTest

Option Name: TARGET_FORMAT Value: F1out

Action: putqueue

Option Name: OPT_TARGET_QUEUE Value: Q2Out

Option Name: OPT_MSG_TYPE Value: MsgTest

NNRie File Layout

Overview

By removing NNRie encryption, you can access and interpret NNRie files with a text editor without having to write an application or utility. In prior releases, the only out-of-the-box access to the Rules database configurations have been through the Rules GUI. Now, with the export files in a readable form, you can write scripts that create Rules components.

Acting on NNRie error reports for imported Rules components had required the use of the Rules GUI and customer support. Now, because the import files

are not encrypted, you can navigate and make changes within the NNRie export files.

WARNING!
The Rules GUI should be used instead of modifying the raw export files. However, this section provides the necessary information for users who are experienced with import/export formats.

Note:
Encryption has been removed from NNFie as well.

NNRie File Layout Guidelines

The following items will help you understand and navigate through a typical NNRie file layout.

- The first line must contain only an R for Rules.
- The second line must convey the version number, for example, 10001, 1.14.1.x.
- Each line after the first one must start with the Rule Component type code defining the layout for the line.
- Commas are the field delimiters. Do not put spaces around commas.
- If a comma is used within a field, it needs to be prefaced with a backslash.
- Components of an application group/message type must be rendered in the following order:
  - application group
  - message type
  - subscription definitions
  - rules definitions
- Subscriptions must be listed before the rules in the file.
- All rules components must start with the definition (for example, Rules 10004 and Subscriptions 10007).

General Format:

R
Version
App1
Msg1 (in App1)
Sub1 (in App1/Msg1)
Action 1 (in Sub1)
Option1 (in Action1)
Permission1 (for Sub1)--only owner and update are listed
Sub2
Action1
Option1
Permission1--owner
Permission2--update
Rule1 (in App1/Msg1)
Permission1 (for Rule1)--only owner and update are listed
Expression (for Rule1)
SubscriptionLink 1 (for Rule1)
Msg2 (in Appl)
}
App2
Msg1 (in App2)
}

Msg2 (in App2)
}

Example of Export File

Field #1	Field #2	Field #3	Field #4	Field #5	Field #6+
Rule Component type code	Application group	Message Type	Rule name or message evaluation type	Subscription name or property/ options for rules	Properties pertaining to rule or subscription

Format Example:

R

10001,4.1.x

10002,sja

10003,sja,InFlat, NEONET_FORMATTER

10007,sja,InFlat,,s1,,1998 / 07 / 14-09:44:43.0,1998 / 07 / 14-09:44:43.0,1

10008,sja,InFlat,,s1,putqueue,1

10009,sja,InFlat,,s1,putqueue,1,OPT_TARGET_QUEUE,1,HitQ

10009,sja,InFlat,,s1,putqueue,1,OPT_MSG_TYPE,2,InFlat

10012,sja,InFlat,,s1,RUL40RUTH,Owner,Granted

10012,sja,InFlat,,s1,RUL40RUTH,Update,Granted

10007,sja,InFlat,,s2,,1998 / 07 / 17-08:58:50.0,1998 / 07 / 17-08:58:50.0,1

10008,sja,InFlat,,s2,putqueue,1

10009,sja,InFlat,,s2,putqueue,1,OPT_TARGET_QUEUE,1,HitQ

10009,sja,InFlat,,s2,putqueue,1,OPT_MSG_TYPE,2,InFlat

10012,sja,InFlat,,s2,RUL40RUTH,Owner,Granted

10012,sja,InFlat,,s2,RUL40RUTH,Update,Granted

10004,sja,InFlat,r1,1,0,0,1

10010,sja,InFlat,r1,PUBLIC,Update,Granted

10010,sja,InFlat,r1,RUL40RUTH,Owner,Granted

10010,sja,InFlat,r1,RUL40RUTH,Update,Granted

10011,sja,InFlat,r1,F1 NOT_EXIST ,1998 / 07 / 17-08:59:19.0,1998 / 07 / 17-08:59:19.0

10013,sja,InFlat,r1,s1

10004,sja,InFlat,r2,1,0,0,1

10010,sja,InFlat,r2,PUBLIC,Update,DenyAll

10010,sja,InFlat,r2,RUL40RUTH,Owner,Granted

10010,sja,InFlat,r2,RUL40RUTH,Update,Granted

10011,sja,InFlat,r2,F1 EXIST ,1998/07/17-08:59:20.0,1998/07/17-08:59:20.0

10013,sja,InFlat,r2,s1

10013,sja,InFlat,r2,s2

Rule Component Types

(10001) Import/Export Version

Example:

10001,1.14.1x

10001	Component type
1.14.1x	Version number

(10002) Application Group

Example:

10002,sja

10002	Component type
sja	Application group

(10003) Message

Example:

10003,sja,InFlat, NEONET_FORMATTER

10003	Component type
sja	Application group
InFlat	Message type name
NEONET_FORMATTER	Evaluation type. This message type refers to a NEONet input format. For 4.1.x this is the only valid evaluation type.

(10007) Subscription

Example:

10007,sja,InFlat,,s2,,1998/07/17-08:58:50.0, 1998/07/17-08:58:50.0,1

10007	Component type
sja	Application group
InFlat	Message type
	Null
s2	Subscription name, which must be preceded and followed by null values delimited by commas.
	Null
1998/07/17-08:58:50.0	Enable date

1998/07/17-08:58:50.0	Disable date
1	Active flag. 1 is active; 0 is inactive.

(10008) Action

Example:

10008,sja,InFlat,,s2,putqueue,1

10008	Component type
sja	Application group
InFlat	Message type
	Null
s2	Subscription name, which must be preceded by a null value delimited by commas.
putqueue	Subscription action name
1	Action sequence number

(10009) Option

Example:

10009,sja,InFlat,,s2,putqueue,1,OPT_TARGET_QUEUE,1,HitQ

10009	Component type
sja	Application group
InFlat	Message type
	Null

s2	Subscription name, which must be preceded by a null value delimited by commas.
putqueue	Action name
1	Action sequence number
OPT_TARGET_QUEUE	Option name
1	Option sequence number
HitQ	Option value

(10010) Rule Permission

Example:

10010,sja,InFlat,r1,PUBLIC,Update,Granted

10010	Component type
sja	Application group
InFlat	Message type
r1	Rule name
PUBLIC	Permission user, may be PUBLIC or individual user.
Update	Permission assigned to user for this instance. (Update / Owner)
Granted	Permissions assigned to PUBLIC for this rule. (Granted / Deny All)

(10011) Rule Expressions

Example:

10011,sja,InFlat,r1,F1 NOT_EXIST ,1998/07/17-08:59:19.0, 1998/07/17-08:59:19.0

10011	Component type
sja	Application group
InFlat	Message type
r1	Rule name
F1 NOT_EXIST	Expression for r1
1998/07/17-08:59:19.0	Enable date
1998/07/17-08:59:19.0	Disable date

(10012) Subscription Permission

Example:

10012,sja,InFlat,,s2,RUL40RUTH,Update,Granted

10012	Component type
sja	Application group
InFlat	Message type
	Null
s2	Subscription name
RUL40RUTH	User, may be PUBLIC or individual username
Update	Permission assigned to RUL40RUTH for this instance. (Update / Owner)
Granted	Permissions assigned to RUL40RUTH for this instance. (Granted / DenyAll)

(10004) Rule

Example:

10004,sja,InFlat,r1,1,0,0,1

10004	Component type
sja	Application group
InFlat	Message type
rl	Rule name

1	Number of arguments
0	This field is not used but must exist.
0	This field is not used but must exist.
1	Active flag. 1 is active; 0 is inactive.

(10013) Rule – Subscription Association

Example:

10013,sja,InFlat,r1,s1

10013	Component type
sja	Application group
InFlat	Message type
r1	Rule name
s1	Subscription name

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 NEONRules. 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 -p <parameter file name>
```

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

Description

The MQSIputdata process reads messages from a file and puts the messages on the queue specified in the parameter file with OPT_APP_GRP and OPT_MSG_TYPE. Optional parameters for put control, such as numRecordsToRead and recordSeparator, allow you to read multiple records from a single file and control how the records are read and committed.

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

Attributes, replytoQ and replyToQmgr, in the MQMD structure can be explicitly set by specifying them in the mpf file. This allows you to bypass the MQSeries Integrator Rules daemon and directs the reply messages to replyToQ.

Operational Assumptions

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

Note:
Error message descriptions and responses are located in *MQSeries Integrator Rules Daemon Error Messages* on page 219.

Parameters

The parameters described in the following tables are used to configure MQSIputdata via a parameter file. The parameters are tunable, meaning that their values can be adjusted to customize control and performance to your environment. The parameter file allows you to enable and disable optional features and to set values of some required features. To view a tunable parameter file example, see *Example* on page 144. A skeleton parameter file is provided with MQSeries Integrator in /examples/MQSIputdata.mpf in your MQSI directory.

The MQSIputdata parameters are divided into the following groups: PutControl, PutMessage, and Put Options. Within the parameter file, the parameters are presented in the same groupings. The group heading must be displayed in the parameter file using square brackets.

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 by put.

Name	Mandatory/ Optional	Description
queueManagerName	Mandatory	Name of the queue manager that owns the queue.
replyToQ	Optional	Name of the queue where the reply message is sent. Parameter is not required when replyToQ is the same as queueName.
replyToQmgr	Optional	Name of the queue manager that owns the replyToQ. Not required when replyToQmgr is the same as queueManagerName.
logFileName	Optional	Name of the log file. If not specified, messages are written to stderr.
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
maxUserDataLength	Mandatory	Maximum message size.
messageCount	Mandatory	Number of messages to put. Default is one (1).

Name	Mandatory/ Optional	Description
transCommitSize	Optional	The integer number of records to be committed at one time. A value of zero (0) indicates commit all records at one time. If the value is zero (0) or more than maxTransCommitSize (an MQSeries queue manager setting), as set for the queue manager, the value defaults to maxTransCommitSize. For additional information about setting maxTransCommitSize, see the MQSeries documentation.
variableLength Record	Optional	Value NO indicates fixed length records are present. Value YES indicates variable length records are present in the file. If YES, recordSeparator is a required parameter and inputFileName must be specified.
segmentation Allowed	Optional	Controls the segmentation of messages being put. Value, YES, allows segmentation. The default value, NO, does not allow segmentation. (Supported only for MQSeries, version 5.0; not supported for messages having header format MQHRF) If value is NO, make sure that the message length does not exceed maxUserDataLength.
recordSeparator	Optional	The ascii character string to be used to determine the end of each record in a file. Parameter is required if variableLengthRecord is set to YES. This parameter is ignored if variableLengthRecord is set to NO.

Name	Mandatory/ Optional	Description
numRecordsToRead	Optional	The integer number of records to load from the file. The default, ALL, indicates that all records are to be loaded.
showStatistics	Optional	Binary value indicating whether or not statistical information should be output. Value of one (1) indicates that the message statistics should be output; zero (0) indicates that the message statistics should not be output.

Put Message

Name	Mandatory/ Optional	Description
format	Mandatory	Populate the format field of the message descriptor with the indicated value.
applIdentityData	Optional	Populate the applIdentityData field of the message descriptor with the indicated value. Use only when the message descriptor contains an applIdentityData field .
putApplName	Optional	Populate the putApplName field of the message descriptor with the indicated value. Use only when the message descriptor contains a putApplName field.

Name	Mandatory/ Optional	Description
applOriginData	Optional	Populate the applOriginName field of the message descriptor with the indicated value. Use only when the message descriptor contains an applOriginName field.
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
includeHeader	Optional	Specifies whether to include the RF header with the inbound message. Value of one (1) indicates that the RF header should be included and Put Options used; 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 (128-byte maximum).
OPT_MSG_TYPE	Optional	Message type associated with the message (128-byte maximum).
User-Defined Option	Optional	User-defined, application-specific option value (128-byte maximum).

includeHeader must be set to a value other than zero (0) for these Put Option settings to be used.

Remarks

MQSeries Integrator uses parameter files of the following structure:

Note:
You can not have trailing whitespace after a group identifier because the file fails to parse correctly. You must have a return immediately following the closing bracket of the group identifier. An example of a group identifier as shown below is: `[Group1]`

```
[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
             .
             .
```

Note:
The spaces before and after the "=" are required.

Example

```
[Put Control]
# Name of the file which contains the message data
# If not specified, an empty/null data file is assumed
inputFileName = putdata.input

# Name of the queue where the message will be put
queueName = myqueue

# Name of the queue manager that owns the queue
queueManagerName = myqmgr

# Name of the queue where the reply message will be sent.
# Comment the following line if the replyToQ is the same as
# queueName
replyToQ = myrtq

# Name of the queue manager that owns the replyToQ. Comment the
# following line if the replyToQmgr is the same as
# queueManagerName.
replyToQmgr = myrtqmgr

# Name of the log file name. Comment the following line if the
# error/warning information is to be logged into stderr.
logFileName = putdata.log

# "log level" used to control message logging to the file.
# Valid settings are:
# 3-log only fatal errors
# 2-log errors, and fatal errors
# 3-log warnings, errors and fatals
```

```
# 4-log informational, warnings, errors, and fatals
logLevel = 0

# Maximum permissible record size in case of variable length
# records. Record size in case of fixed length records.
maxUserDataLength = 40000

# Number of times each message to put in the queue. An integer
# value must be specified.
messageCount = 1

# Transaction commit size, 0 indicates commit all the records
# once
transCommitSize = 30

# A value YES indicates variable length records are present in
# the file. A value NO indicates fixed length records are
# present in the file.
variableLengthRecord = YES

# segmentationAllowed, YES if allowed, NO if not allowed
segmentationAllowed = YES

# Record separator string (ascii). Used in conjunction with
# variable length records to indicate the end of record. Its
# value must be specified, if variableLengthRecord is YES.
# Otherwise its value is ignored.
recordSeparator = xxxx

# Number of records to load from the file, ALL if all records
# are to be loaded
numRecordsToRead = ALL

# Binary value indicating whether of not statistics information
# should be output. 1 indicates yes, 0 indicates no.
showStatistics = 1

[Put Message]

# Populate the format field of the message descriptor with this
# value.
format = MQHRF
```

```
# Populate the ApplIdentityData field of the message descriptor
# with this value. The following line to be commented if no
# ApplIdentityData field is present in the message descriptor.
applIdentityData = xxx

# Populate the PutApplName field of the message descriptor with
# this value. The following line to be commented if no
# PutApplName field is present in the message descriptor.
putApplName = MQSIputdata

# Populate the ApplOriginData field of the message descriptor
# with this value. The following line to be commented if no
# ApplOriginData field is present in the message descriptor.
applOriginData = xxx

# Populate the expiry field of the message descriptor with this
# value.
expiry = -1

# Populate the persistence field of the message descriptor with
# this value.
# Valid values for persistence
#          MQPER_PERSISTENT            1
#          MQPER_NOT_PERSISTENT        0
#          MQPER_PERSISTENCE_AS_Q_DEF  2
persistence = 0

# Populate the message type field of the message descriptor
# with this value.
# Valid values for message type:
#           MQMT_REQUEST               1
#           MQMT_REPLY                 2
#           MQMT_REPORT                4
#           MQMT_DATAGRAM              8
messageType = 8

# Specify whether or not to include the RF header
# with the inbound message 1 = yes, 0 = no
includeHeader = 1

# Specify how to populate the MQRFH.Format field.
```

```
# This parameter only takes effect if includeHeader == 1.
dataFormat = MQSTR

[Put Options]

# This group defines the options which will be attached to the
# to the message before it is sent.  The parameters in this
# group only take effect if includeHeader = 1.
OPT_APP_GRP = mqsiAG
OPT_MSG_TYPE = mqsiIF
```

MQSIgetdata

Syntax

```
MQSIgetdata.exe -p <ParameterFileName>
```

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

Description

The getdata program retrieves messages and options from an MQSeries Integrator Rules Daemon input queue and puts the message to an output file specified in the tunable parameter file, MQSIputdata.mpf.

Operational Assumptions

- Queue manager is up and running.
- Queues have been created. The MQSI getdata program expects that the queue name defined in the tunable parameter file exists, is enabled, and has messages on it.

Note:
Error message descriptions and responses are located in *MQSeries Integrator Rules Daemon Error Messages* on page 219.

Parameters

Get Control

Name	Mandatory/ Optional	Description
outputFileName	Optional	Name of the file to contain messages. Default is stderr. Required if outputToFile is set to 1.
queueName	Mandatory	Name of the queue holding the messages.
queueManagerName	Mandatory	Name of the queue manager that owns the queue.
maxUserDataLength	Mandatory	Indicates the maximum message size that the application can accept.
logFileName	Optional	Specifies the name of the log file for error / warning / information reporting. If not specified, logging defaults to stderr.
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
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.

Name	Mandatory/ Optional	Description
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	Optional	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.
commitSize	Optional	The integer number of records to be committed at one time. Zero (0) value causes all records to be committed at one time.
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 exits. getTimeout values are measured in milliseconds.
showStatistics	Optional	Shows statistics about messages taken of the queue. 1 indicates that this feature is enabled; zero indicates that this feature is disabled.
outputToFile	Optional	Indicates whether or not an output should be sent to a file. Value of zero (0) indicates that the output should be sent to stderr. Value of one (1) indicates that output should be sent to the file specified by outputFileName .

Name	Mandatory/ Optional	Description
showDescriptor	Optional	Indicates whether or not the message descriptor data should be output. 1 indicates that the message descriptor data should be output; zero 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 indicates that the message data should not be output.
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 indicates that the messages should not be rolled back.

Remarks

MQSeries Integrator uses parameter files of the following structure:

Note:
You can not have trailing whitespace after a group identifier because the file fails to parse correctly. You must have a return immediately following the closing bracket of the group identifier. An example of a group identifier as shown below is: `[Group1]`

```
[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
              .
              .
```

Note:
This spaces before and after the "=" are required.

Example

```
[Get Control]
# Name of the file to put the message in.
outputFileName = getdata.output

# Name of the queue to get the message from.
queueName = myqueue

# Name of the queue manager that owns the queue.
queueManagerName = myqmgr

# Maximum message size that the application can get.
maxUserDataLength = 40000

# Name of the log file. Comment the following line if the
# error/warning information is to be logged into stderr.
logFileName = getdata.log

# "log level" used to control message logging to the file.
# Valid settings are:
# 3-log only fatal errors
# 2-log errors, and fatal errors
# 3-log warnings, errors and fatals
# 4-log informational, warnings, errors, and fatals
logLevel = 0

# ID of the message to get.  If this value is not defined
# and correlID is not defined, the application gets the
# next available message from the queue.  Notice that this
# field uses an encoded hex representation for the messageId.
messageId = 414D51205141514D20202020202020202034EA17130000030D

# Correlation ID of the message to get.  If this value is
# not defined and messageID is not defined, the application
# gets the next available message from the queue.  The
# correlID field uses an encoded hex representation of a
# binary value.
correlId =

# Maximum number of messages to get.  The application will
# run until messageCount messages have been dequeued or
# until the queue is empty.
```

```
messageCount = 3000

# Transaction commit size, 0 indicates commit all the
# records once
commitSize = 0

# Maximum amount of time to wait for a message to arrive
# before the application reports a queue empty and exits.
# As of MQSeries version 5, the units of this timeout value
# are milliseconds.
getTimeout = 0

# The following entries are binary attribute indicators
# 1 indicates that the feature should be enabled.  0
# indicates that the feature should be disabled.
# Show statistics about dequeued messages.
showStatistics = 1

# Should the output be sent to a file.  0 indicates that
# output should be sent to stderr.
outputToFile = 1

# Should the message descriptor data be output.
showDescriptor = 1

# Should the message data be output.
showData = 1
```

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 NEONFormatter.

Operational Assumptions

- A complete and valid installation of MQSeries Integrator release 4.0 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 NEONFormatter 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.
-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 the message associated with the rule. When the utility completes processing, it displays whether the rule will 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.

The Rules Engine Executable

The Rules Engine executable 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 Rules Engine is a daemon that performs rule evaluation against a specified message and attempts to execute actions for rules that evaluate to true. Users can define rules using the GUIs (see ***Using NEONet***) or by using the Management APIs (see the ***Programmer's Reference for the Rules APIs***). Application programmers can use the Rules APIs to evaluate rules (see the ***Programmer's Reference for Rules APIs***).

Rules Engine Processing

The Rules Engine daemon is built on top of the Rules APIs and performs these procedures, in this order:

1. Polling
2. Message processing
3. Rules Caching
4. Action execution
5. Failure processing

Polling

Polling of a NEONet queue occurs based on the "wait time" parameter specified by the Rules Engine command parameter. When message processing, subscription execution, and failure processing are complete, the polling process begins again. Polling performs these functions:

- Prior to polling, or reading a NEONet queue, the transaction boundary is defined to ensure that no data is lost.
- If the queue is read successfully and a message is retrieved, processing of the message begins. If the queue is empty, the Rules Engine still attempts to read the queue.

- If the queue is not read successfully or the queue is empty, the transaction is rolled back and the transaction end is defined.

The Rules daemon checks the input queues consecutively, rather than checking a single queue until it is empty. If it is specified, the Rules daemon also checks the reload queue each pass before going through the rest of the queues; starting with the queue after the previously read queue.

Note:
When running the Rules daemon, no other process should drain the input queue(s). This destroys guaranteed delivery / guaranteed sequence and may cause long waits and possible deadlocks.

Timestamps

Timestamps make it easy to determine the timing of events in the Rules daemon if there are problems processing messages. Timestamps are in a universal (international) format and are added to the log file, if it exists, every time a message is processed. Timestamps in the Rules daemon are in ISO format of the local date / time: YYYYMMDDHHMMSS.

Message Processing

Message processing evaluates the message against the currently defined rule set for the application group / message type pair. NEONet Formatter is called to deconstruct (parse) the input message into component parts (fields). NEONet 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 fails (and the message is sent to the Failure Queue). Only active subscriptions are retrieved for hit rules. If there are no active subscriptions in a rule set (application group / message type), the load fails (and the message is sent to the Failure Queue). If none of the hit rules have active subscriptions, the first call of the

getsubscription() API returns nothing (and the message is sent to the NoHitQ Queue).

Rules Caching

When users change data within a rule or rule set that is specified by an Application Group / Message Type name pair, they need to signal a running Rules Engine instance to reload the changes into memory. Users can also change data for a single subscription without reloading the entire ruleset.

Ruleng can be configured to check a queue for reload notification messages (see *Using the Rules Engine* for details about configuring Ruleng). Notification messages are typically empty and have five options -- OPT_APP_GRP and OPT_MSG_TYPE set to the application group and message type indicating which rule set to reload, OPT_RELOAD set to TRUE indicating to the Rules Engine to reload the specified rule component, or the OPT_COMPONENT_ TYPE set to SUB or MSG, and if reloading a subscription, the OPT_SUB_ NAME set to the appropriate subscription name. If the message contains data, the message will process after the cache is reloaded.

You must decide whether to create a new queue for notification notices or use an existing queue (such as the queue the Rules Engine uses to get messages).

NNRSignalReload is an executable provided to put an empty message onto a queue with the correct Application Group / Message Type information, OPT_RELOAD set to TRUE, and OPT_COMPONENT_TYPE set to MSG or SUB. Use putdata to add the appropriate reload options to a data message put on a 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 (no rules evaluate to true) or no active subscriptions exist for the hit rules, the message is routed to the No Hit Queue.

By default, if the output queue (or no hit queue) is full or writing is disabled on it, the Rules daemon rolls back and waits for the queue to become available before processing additional messages from the input queue. The wait time is the same as the wait time used for checking the input queue for messages (the -w command line parameter). This default behavior puts the

Rules daemon on hold so the input queue can become full while waiting for the output queue to become writable. If logging is turned on, a note in the log file notifies the user that the output queue is full or disabled and the system is waiting. The user must then drain and enable the output queue so that processing can continue. If the output queue does not exist or has any problem other than the queue is full or disabled, the original message is placed on the failure queue. If the daemon is unable to put to the failure queue, the daemon rolls back and the rules engine stays on that message until it can write to the failure queue after the wait time.

To override the default waiting, the user must set the queue option OPT_NO_WAIT to TRUE when putting the original message on the input queue. This option causes the message to be put to the failure queue if the output or no-hit queue is full or disabled. NEONet's putdata utility takes -n parameter which sets this option to TRUE when putting a message to an input queue. The Rules daemon passes on any options that are set in the input message when it does a putqueue (overriding the OPT_MSG_TYPE based on the subscription option), except for OPT_RELOAD. Therefore, if OPT_NO_WAIT is set on the input message, the output queue has that option set as well. To explicitly set the Rules daemon to wait, the OPT_NO_WAIT option should be set to FALSE.

If at any time during subscription execution there is a failure, 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 Rules Engine are Reformat, Put Queue, and ReEval. Other actions defined require users to write their own rules engine daemon to process those actions. The Rules Engine does not execute generic actions. Subscriptions are not executed in a specific order. If messages must be output in a certain order, the Put Queue actions must be in a single subscription. Each subscription starts with the original message.

Reformat

The Reformat action takes a message with an input format and re-formats the message to a message adhering to the specified output format. The Reformat action requires both an input and output format as options. Formatter APIs are called to perform the reformat of messages. If using Rules Management APIs to add the Reformat action, the action name should be reformat with the option name: INPUT_FORMAT and TARGET_FORMAT (with case as specified). If consecutive reformat actions occur in a subscription, the input message to the second reformat is the output of the first reformat.

User Exit Constraints

The same database must be used for Messaging and Queuing, Formatter, and Rules if database modifications are made in a user exit.

Furthermore, to insure database recovery in the event of a failure, user exits should only perform updates and inserts to the same database used by Messaging and Queuing, Formatter, and Rules. If the Messaging and Queuing, Formatter, and Rules data reside in two different databases, the modifications made during User Exits will lead to unpredictable behavior and a possible loss of data integrity.

No Database Commits in a User Exit

To preserve the integrity of the transaction layer, users should not explicitly commit database changes made in User Exits. If a single database is used for Queuing and Formatter/Rules, then database modifications made during User Exits will get committed by the Rules Engine when the transaction is completed.

PutQueue

The Put Queue action takes a message and puts it onto a specified destination NEONet queue and sets the message type option, if it exists, to the message format type specified. The Put Queue action requires a destination NEONet queue name as an option. The queue name must exist in the NEONet database. The message placed on the queue is either the original message or the result of the last previous reformat action in the same subscription. NEONet Messaging and Queuing APIs are called to perform the Put Queue operation. The Put Queue does not perform formatting. If using Rules

Management APIs to add the Put Queue 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 (MQDirect) version of NEONet, the Rules Engine uses MQSeries to put to the output queues.

WARNING!
If a subscription does not include the Put Queue 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 Put Queue.

Note:
While the Reformat and Put Queue subscription options are the only actions that can be performed by the Rules Engine, the NEONet Rules APIs allow any number of actions and associated options. An application programmer can use NEONet 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 NEONet Rules APIs.

ReEval

The ReEval action

The ReEval actions are performed after all other actions are done. The list of ReEval actions is then traversed: the given message buffer is parsed and evaluated, and the resulting actions are performed. If a re-eval action is encountered at this time, the item is added to the list of re-evals to be performed as was done previously. If the OPT_EVAL_ORDER is CHILD_FIRST, the child re-eval will be done before any sibling re-evals. If the OPT_EVAL_ORDER is SIBLING_FIRST or not specified, the child re-eval will be placed at the end of the list re-evals.

Shutdown Messages

The Shutdown message supplies a graceful way to shutdown the Rules daemon. A shutdown message has the option OPT_SHUT_DOWN set to TRUE. If the message has no data, the daemon shuts down gracefully. If data exists on the message, message processing is done before the daemon is

shutdown. If the rules do not hit, the message goes to the NoHit Queue before the Rules daemon shuts down. If there is a failure, the Rules daemon shuts down after the message is put on the Failure Queue. If an output queue is full or disabled and OPT_NO_WAIT is NOT set, the message rolls back and the daemon does not shut down until the message can be written to the output queue.

The get from the queue commits so this message is not read again when the Rules daemon is brought up. The user can put a shutdown message on any queue that is read by the daemon. For more information, See *NNRSignalShutdown Utility* on page 192. and See *putdata* on page 181.

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 NEONet Failure Queue specified in this process. Using the Rules Engine daemon, you can write a process to manage the messages in the Failure Queue.

The NEONet Rules Engine can be configured to set the OPT_ERR_CODE and / or the OPT_ERR_MSG queue option each time a message is put to the NEONet Failure Queue. Both options are intended to help users determine why the Rules Engine sent the message to the Failure Queue. The OPT_ERR_CODE queue option value will indicate which subsystem (i.e., Messaging and Queuing, Formatter, or Rules) encountered the failure and provide the error code number. (For a complete listing of the NEONet error codes, see *Programmer's Reference* section Error Codes, Names and Messages. The following prefixes will be added to the error code number to indicate the subsystem:

- NNF (NEONet formatter)
- NNQ (NEONet Messaging and Queuing)
- NNR (NEONet Rules)

The OPT_ERR_MSG queue option value will provide the error message that corresponds to the error code number.

Rules Engine Daemon Error Codes

Code	Error Name	Explanation	Response
-10000	RULENG_INVALID_ PUT_QUEUE_ ACTION_ERR	Putqueue action contains invalid or missing OPT_TARGET_QUEUE option name and/or value.	Correct the options in the Putqueue action.
-10001	QUEUE_CREATION_ FAILURE_ERR	Failure creating queue specified in the Putqueue actions's OPT_TARGET_ QUEUE option.	Correct the options in the Putqueue action.
-10002	QUEUE_ INITIALIZATION_ FAILURE_ERR	Failure to initialize queue specified in the Putqueue action's OPT_TARGET_ QUEUE option.	Verify that the specified queue exists.
-10003	RULENG_INVALID_ REFORMAT_ACTION _ERR	Reformat action is missing a value for the INPUT_FORMAT <or> the TARGET_FORMAT value is missing.	Correct the options in the Reformat action.

Message Routing

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

- If no subscription actions are executed the message is routed to the No Hits 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.

- The Rules daemon will wait if the output queue or no-hit queue is full or disabled, unless the incoming message had the OPT_NO_WAIT option set to TRUE after the wait time.

Configuration Prior to Using the Rules Engine Daemon

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

Queues

The NEONet Rules Engine uses input and output NEONet queues. Input queues are the queues specified by the -q parameter of the Rules Engine. Output queues are: failure queue; no hits queue; and any queue(s) specified by any Put Queue action.

To have a message successfully evaluated by the Rules Engine daemon, the input message must have these two options set:

```
OPT_APP_GRP
OPT_MSG_TYPE
```

OPT_APP_GRP assigns the message to an application group and must match the application group name in the Rules database. The OPT_MSG_TYPE must match the message type in rule definitions and the input format name in the format definitions. These two options on the message allow the Rules Engine to evaluate the message against its rules and only its rules. These options can be set using either the M&Q SetOpt and SetOptSet APIs or the NNHPutMsg API. The queue may also have default OPT_APP_GRP and OPT_MSG_TYPE options defined. If these options do not exist on the message, the default OPT_APP_GRP and OPT_MSG_TYPE defined for the Rules daemon will be used, if it is supplied. If the options are not set the evaluation will not occur and failure processing will occur. (Refer to the ***Programmer's Reference for High-Level APIs and Messaging and Queuing APIs*** for information on using these APIs.)

To override the default waiting, the user must set the queue option "OPT_NO_WAIT" to "TRUE" when putting the original message on the input queue. This option will cause the message to be put to the failure queue if the output or no-hit queue is full or disabled. If the -n parameter is specified when invoking the putdata utility, then the OPT_NO_WAIT option will be set to TRUE when putting a message to a queue. The Rules daemon passes on any options that are set in the input message when it does a putqueue. Therefore, if OPT_NO_WAIT is set on the input message, the output queue will have that option set as well. To explicitly set the Rules daemon to wait, the OPT_NO_WAIT option should be set to FALSE.

Note:
The OPT_MSG_TYPE specified on the input message will be overridden by the OPT_MSG_TYPE option specified in the putqueue action.

Rules

The Rules daemon checks incoming messages for the reload queue options. The Rules daemon calls the Rules reload API for the component specified in the OPT_COMPONENT_TYPE queue option. The Rules daemon reloads the entire rule set (defined by application group / message type) if it encounters messages with either the old OPT_RELOAD_RULE_SET option set to TRUE or if the OPT_COMPONENT_TYPE option set to MSG and the OPT_RELOAD option is set to TRUE. The Rules daemon reloads an individual subscription if it encounters a message with OPT_RELOAD set to TRUE, OPT_COMPONENT_TYPE set to SUB, and OPT_SUB_NAME set to the appropriate subscription name. The subscription may have been added to, updated in, or deleted from the database and the corresponding change is done in the cache. If the reload message contains data, the reload is performed and then the message is evaluated.

WARNING!
Unless Reload messages are used, the NEONet Rules daemon is not dynamic with respect to rule definition (this also includes subscription definition). Only rules defined prior to the Rules daemon startup are used. Any rules added or changed after the Rules daemon startup are not used until the reload message is read.

Formats

WARNING!
All NEONet 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 NEONet formats needed during a reformat action must be entered and saved prior to starting the NEONet Rules daemon.

Note:
For information about entering queues, rules, and formats, refer to ***Using NEONet*** and the ***Programmer's Reference*** documents.

Running the Rules Engine

The following UNIX and NT Services sections provide information on how to run the ruleng.

Running ruleng on UNIX

To start the Rules engine, either pass in the commands on the command line or follow the prompt for the parameters.

To close the Rule engine, send a shutdown message. To send a shutdown message, complete the following steps:

1. Run:

   ```
   NNRSignalShutdown
   ```

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

   ```
   putdata -d putdata.mpf
   ```

Running ruleng as an NT Service

You can run ruleng as a Service under Windows NT 3.5 or higher (NT 4.0 is recommended). In addition, Messaging and Queuing daemons, XMIT and RECV can also be run as NT Services. See *Running neonXmit and neonRecv as NT Services* on page 31.

The benefits of running ruleng as a service include the ability to:

- Start, stop, and pause the service from the service GUI provided by NT, or from the command line.
- Run multiple services at one time.
- Schedule a service to start automatically when a machine is unattended.

Naming Your Services

Before installing your services, you must determine a unique name for each service to be installed. The service name points any service commands to the appropriate messaging and queuing engine or the appropriate rules engine.

Consider the following:

- Uniquely name each entity in relation its function or component rather than relying on case sensitivity to distinguish them. To prevent potential database conflicts, use case sensitivity as a readability factor but not as the sole differentiation between names. While TestCase, TESTCASE, and testcase are each considered as unique names in a case-sensitive database, they are duplicates within a database that is not case sensitive.
- Choose short names that are meaningful to your systems and organization.
- Capitalize each word in a name to make it more readable and to serve as a consistency standard (such as *TestImport*.)

Notes:
The service names and corresponding executables are stored in your system's registry. It is not recommended that the registry executables be changed or modified.

Installing NEONet NT Services

Services are named and installed from the command line and must be installed before you can run them. The following install syntax handles the naming of your service and its installation.

```
ruleng -install <your service name> <options for ruleng>
```

When the installation is complete, a confirmation message appears. If installation is unsuccessful, an error message appears.

To verify that the service is installed and is available, open the NT services window, as described in the following section. To further check the status of the service and display the default parameters / options, use the following command:

```
ruleng -info <your service name>
```

Example:

To install a service named RulesWest for ruleng, type:

```
ruleng -install <RulesWest>
```

Managing Services from the NT GUI

From the Windows NT desktop, select Start→Settings→Control Panel. Double-click Services in the Control Panel window.

The NT Services window appears. Installed Services are listed in the window. For each Service, its current status and startup method is listed.

Running a Service

1. From the NT Services window, highlight the Service you want to start. Click Start. The Status changes to Started.
2. To Stop or Pause a Service, highlight the Service and click Stop or Pause. The Status will reflect the change.

Scheduling Automatic Startup for a Service

Services can be scheduled to start automatically according to parameters you set. Highlight the service you want to schedule and click Startup.

1. Choose the Startup Type.

   Automatic: Service runs automatically when the system starts. The Service will start only if the computer has 12MB or more of random access memory (RAM).

Manual: Service runs only when started by a user or a dependent service. Service remains running until it is stopped, even if the user that started the service has logged off the system.

Disabled: Service is disabled and will not start.

2. Identify the Log On As parameters.

   System Account: The Service logs on to a system account versus a user account. Most Services log on to a system account.

   This Account: The Service logs on to a specific user account with corresponding password. Click the browse button to specify a user account, and then type the password for the user account in both the Password and Confirm Password boxes.

   To provide a user interface on the desktop that can be used by whoever is logged in when the service is started, select the Allow Service to Interact with Desktop check box.

3. Click OK.

Managing Services from the Command Line

The command line can provide more detailed descriptions of errors when they occur.

Notes:
The GUI does not automatically refresh after a command line operation is performed affecting a service. Close and reopen the GUI to view the change to the service.

Syntax

```
ruleng <service option> <service name> <standard option
(optional, as needed)>
```

Service Options

Service Option	Description
`-install`	Installs the service with the <standard option> used as startup defaults.
`-remove or -uninstall`	Removes the service.
`-start`	Starts the service. The <standard options> override default startup parameters.
`-stop`	Stops the service.
`-pause`	Pauses the service if it is running.
`-continue`	Continues the service if it is paused.
`-info`	Prints service status and current default parameters.
`-parameters`	Installs <standard options> as the default startup parameters the next time the service is started. To change parameters for one time only, use the GUI.
`-? or -help`	Displays service usages and printing of usage text.

Examples

Starting a Service in ruleng

To start a service named RulesWest in ruleng with no options:

```
ruleng -start RulesWest
```

By starting the service from the command line versus through the GUI, the default startup options will be overridden by no options in this command. This occurs for this service start only because the default startup parameters are not permanently changed by the command line start.

To Set the Default Parameters for ruleng Startup

To set parameters for the next time the RulesWest service is started for ruleng:

```
ruleng -parameters RulesWest -s MySes -q Q1 -F FailQ -N NoHitQ
```

The next time ruleng starts, these parameters will be used, overriding the parameters set for ruleng for this startup only.

To Uninstall the Service from ruleng

To uninstall RulesWest as a service for ruleng, type:

```
ruleng -uninstall RulesWest
```

RulesWest will no longer be available as a service and will not appear on the NT Services window.

Using the Rules Engine

ruleng

Syntax

```
ruleng -s <queue session name> [-r <rule session name>] [-l
<logfile name>] [-w <wait time seconds>] -F <failure queue
name> -N <NoHitQ queue name> [-v] [-R <queue to check for
reload notification>] [-p <wait time for empty reload queue>]
[-e] [-E] [-A <default application group name>][-M <default
message type name>] [-L]
```

Configuration File

Before running the Rules Engine, 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. Specifically, a session name that is the same as the value specified by the -s parameter for queues below; and, a session name that is the same as the -r parameter below.

Parameters

Name	Mandatory/ Optional	Description
-s <queue session name>	Mandatory	The queue session name corresponding to the identifier in the NEONet configuration file. This session should connect to a database where the queues are defined.
-r <rule session name>	Optional	The rules session name corresponding to the identifier in the NEONet configuration file. This session should connect to a database where the rules and formats are defined. If this parameter is not present, the queue session will be used. If the rules or queue session names are the same, only one database session will be used. This must be done if user exits that change the database are present.
-l <log file name>	Optional	The Rules Engine will log errors to this file. To optimize performance, the default is to do no logging. (This includes error logging.)
-w <wait time seconds>	Optional	Specifies the time, in seconds, between queue reads. Default is three (3) seconds.
-F <failure queue name>	Mandatory	Indicates the queue to route messages that could not be evaluated or failed during subscription execution. If, at any time during rules evaluation or subscription execution, there is a failure, messages will be put on the failure queue identified by this parameter.

Name	Mandatory/ Optional	Description
-N <no hit queue name >	Mandatory	The Rules Engine puts messages that evaluated to false (if every rule in the Application Group / Message Type was false for the message), or messages that do not have active subscriptions onto the no hit queue.
-v (verbose)	Optional	Directs output to both the specified log file and the user's screen. To optimize performance, the default is to do no logging to the file or screen.
-R <queue to check for reload notification>	Optional	Specifies what queue to check for reload notifications. If the reload queue isn't one of the input queues, polling will only occur at the interval (in seconds) set by the -p option.
-p <wait time for empty reload queue>	Optional	Number of seconds to wait after processing all messages before polling for messages again. (Default is set to the time specific by the -w parameter.)
-e	Optional	Directs the Rules Engine daemon to set the OPT_ERR_CODE queue option on all messages put to the Failure Queue.
-E	Optional	Directs the Rules Engine daemon to set the OPT_ERR_CODE and the OPT_ERR_ MSG queue options on all messages put to the Failure Queue.
-A <default application group name>	Optional	Specifies what application group name to use when a message without the OPT_APP_GRP queue option is read by the Rules Engine daemon. "default" is the default.

Name	Mandatory/ Optional	Description
-M <default message type name>	Optional	Specifies what message type name to use when a message without the OPT_MSG_TYPE queue option is read by the Rules Engine daemon. “default” is the default.
-L	Optional	Indicates when to reload messages. If this option is not set, a flag to reload the appropriate item is set when a reload message is received and is reloaded when the eval() is called. NEON recommends this selection. If this option is set, the data reloads every time a reload message is received rather than wait for it to be evaluated. For subscription reloads, this requires the daemon to load the entire set of rules for an application group / message type if it has not yet been loaded into cache.

Note:
If ruleng is run without any command line parameters, it will prompt the user for the required information.

Example

Command line:

```
ruleng -r rules -s queues -q inputQ -N NoHitQ -F FailureQ
-w 10 -v
```

sqlsvses.cfg (Sybase)

```
rules:MyServerName:MyUserName:MyPassword:MyRulesDB:
queues:MyServerName:MyUserName:MyPassword:MyQueuesDB:
```

sqlsvses.cfg (Oracle)

```
rules:MyServerName:MyUserName:MyPassword:
queues:MyServerName:MyUserName:MyPassword:
```

Note:
For Oracle, when entering the server name, do not use @.

NNRSignalReload

NNRSignalReload is an executable provided to put an empty message onto a queue with the options OPT_APP_GRP set to the application group, OPT_MSG_TYPE set to the message type, OPT_SUB_NAME set to the subscription name (for a subscription), OPT_COMPONENT_TYPE set to the component to reload (MSG or SUB), and OPT_RELOAD set to TRUE. See *Rules Caching* on page 161.

To signal a single subscription reload, users must provide a value for each of the mandatory input parameters, the subscription to reload, and SUB as the component type. An error and usage statement is displayed if the -S parameter is not selected when the component type is set to SUB.

To signal a reload of an entire rule set, MSG must be the component type.

Note:
NNRSignalReload reloads an entire rule set if the -S and -C parameters are not used. However, the -S parameter is ignored when MSG is specified as the component type.

Syntax

```
NNRSignalReload -a <application group> -m <message type> [-S
<subscription name>] -q <queue name> [-s <session name>] [-C
(MSG | SUB)] [-v]
```

Parameters

Name	Mandatory/ Optional	Description
-a <application group>	Mandatory	Sets an option on the OPT_APP_GRP queue to the specified application group. If the process is unable to set this option, a failure occurs and the process terminates.

Name	Mandatory/ Optional	Description
-m <message type>	Mandatory	Sets an option on the OPT_MSG_TYPE queue to the specified message type. If the process is unable to set this option, a failure occurs and the process terminates.
-S <subscription name>	Optional	Sets an option on the OPT_SUB_NAME queue. If the process is unable to set this option, a failure occurs and the process terminates.
-q <queue name >	Mandatory	Sets the NEONet queue name where the reload message is added. This should match an input queue for ruleng or the name specified with the -R option for ruleng.
-s <session name>	Optional	Session name tag from the sqlsvses.cfg file. The default session name is input.
-C <component type>	Optional	Assigns a value to the OPT_ COMPONENT_TYPE queue option based on the component type value specified in this argument. The supported values include MSG and SUB.
-v (verbose)	Optional	Directs output to both the specified log file and the user's screen. To optimize performance, this defaults to no logging to the file or screen.

Example NNRSignalReload calls

Case 1

```
NNRSignalReload -a TestApp -m TestFmt -v -s rules -q InQ -r -C
SUB -S TestSub

Uses "rules" session to connect to database. Puts empty message
on "InQ" queue with OPT_APP_GRP and OPT_MSG_TYPE set.
```

```
OPT_RELOAD will be set to TRUE. OPT_COMPONENT_TYPE will be SUB
and OPT_COMPONENT_NAME will be TestSub. This signals the Rules
Daemon to reload the subscription.
```

Case 2

```
NNRSignalReload -a TestApp -m TestFmt -v -s rules -q InQ -r -C
MSG

Uses "rules" session to connect to database. Puts empty message
on "InQ" queue with OPT_APP_GRP and OPT_MSG_TYPE set.
OPT_RELOAD will be set to TRUE. OPT_COMPONENT_TYPE will be MSG
and OPT_COMPONENT_NAME will be TestFmt. This signals the Rules
Daemon to reload the rule set. The -C MSG is not necessary for
this call
```

Testing Rules

Rules Test Programs

The putdata, getdata and ruletest programs are provided for testing the Rules Engine program. These test programs are explained in this section.

putdata and getdata

The putdata program can be used to put data onto a 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 Rules daemon output queue.

putdata

Syntax

```
putdata -i <input filename> -a <application group> -m <message
type> [-v] [-n (set no wait)] [-r (set reload option) [-C (MSG
| SUB)] [-S <subscription name>] ] [-q <queue name>] [-s
<session name>] [-d (set shutdown)]
```

Description

The NEONet putdata process reads a message from a file and puts the message on the specified queue name if the -q parameter is used or on the queue named RulesIn with the OPT_APP_GRP, the OPT_MSG_TYPE, and possibly the OPT_NO_WAIT, OPT_RELOAD, or OPT_SHUT_DOWN options set. The RulesIn queue is a possible input queue for the NEONet Rules Engine Daemon and should be specified as such in the ruleng as "-q RulesIn". The OPT_NO_WAIT option causes the Rules daemon to put messages on the Failure Queue if the output queue is full or disabled. The OPT_RELOAD option causes the Rules daemon to reload its cache before evaluating the message. The OPT_SHUT_DOWN option causes the Rules daemon to shut down after processing the message.

This process sets the two options on the message that the NEONet Rules daemon expects, specifically the application group and message type. It may also set the other specified options.

To reload the entire rule set, the -r parameter must be set. To reload a single subscription, users must provide a value for each of the mandatory input parameters, the subscription to reload, and SUB as the component type to reload. An error and usage statement is displayed if the -S parameter is not set when the component type is set to SUB.

Note:
The -S parameter is ignored when MSG is specified as the component type.

Operational Assumptions

- A complete and valid installation of NEONet must exist prior to running the NEONet Rules Engine Daemon. The database must also be running in a stable state prior to running the NEONet putdata process.
- Both the putdata and getdata programs require a connection to a database containing NEONet queuing data.
- The NEONet putdata process expects that the specified queue exists, and is enabled, and is defined in the -q parameter in the ruleng run.

Configuration File

Before running this executable, you must 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.

The session name in the sqlsvses.cfg file is used by the Rules Engine to locate the appropriate line from which to retrieve connection data. The putdata program expects to have a session name of “input” (unless otherwise specified). Using this connection data, the Rules Engine test programs can make a connection to the appropriate database.

Example Configuration File

```
input:MyServerName:MyUserName:MyPassword:MyDB
```

Parameters

Name	Mandatory/ Optional	Description
-i <input filename>	Mandatory	The input file from which putdata 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 NEONet putdata program sets an option on the queue called OPT_APP_GRP. If the process is unable to set this option, a failure occurs and the process terminates.
-m <message type/format name>	Mandatory	The NEONet putdata program sets an option on the queue called OPT_MSG_TYPE. If the process is unable to set this option, a failure occurs and the process terminates.
-v (Verbose)	Optional	The NEONet putdata program will log to the screen if this optional parameter is set. The process defaults to no logging.

Name	Mandatory/ Optional	Description
-n (Set NoWait)	Optional	If -n exists, the OPT_NO_WAIT option will be set to TRUE on the specified queue when the message is put.
-r (Set Reload)	Optional	Sets the queue option OPT_RELOAD_ RULE_SET to TRUE on the specified queue.
-C <component type>	Optional	Assigns a value to the OPT_COMPONENT_ TYPE queue option based on the component type value specified in this argument. The supported values include MSG and SUB.
-S <subscription name>	Optional	Sets an option on the queue called OPT_SUB_NAME. If the process is unable to set this option, a failure occurs and the process terminates.
-q <queue name >	Optional	Sets the NEONet queue name where the message will be added. This should match an input queue for ruleng or the name specified with the -R option for ruleng. The default queue name is RulesIn.
-s <session name>	Optional	Session name tag from the sqlsvses.cfg file. The default session name is input.
-d (Set ShutDown)	Optional	putdata puts the message on the named queue with the option OPT_SHUT_DOWN set to TRUE.

Example putdata calls

Case 1

```
putdata -i inputfile.txt -a TestApp -m TestFmt -v

Uses default session input to connect to database. Puts message
from inputfile.txt onto "RulesIn" queue with OPT_APP_GRP and
OPT_MSG_TYPE set. NOTE: The message will contain all data from
```

inputfile.txt until the end of the file.

Case 2

```
putdata -i inputfile.txt -a TestApp -m TestFmt -v -n
```

Uses default session input to connect to database. Puts message from inputfile.txt onto "RulesIn" queue with OPT_APP_GRP and OPT_MSG_TYPE set OPT_NO_WAIT is also set to TRUE.

Case 3

```
putdata -i inputfile.txt -a TestApp -m TestFmt -v -s rules -q
InQ -r -C SUB -S TestSub
```

Uses "rules" session to connect to database. Puts message from inputfile.txt onto "InQ" queue with OPT_APP_GRP and OPT_MSG_TYPE set. OPT_RELOAD will be set to TRUE. OPT_COMPONENT_TYPE will be SUB and OPT_COMPONENT_NAME will be TestSub. This signals the Rules Daemon to reload the subscription before evaluating this message.

Case 4

```
putdata -i inputfile.txt -a TestApp -m TestFmt -v -s rules -q
InQ -r -C MSG
```

Uses "rules" session to connect to database. Puts message from inputfile.txt onto "InQ" queue with OPT_APP_GRP and OPT_MSG_TYPE set. OPT_RELOAD will be set to TRUE. OPT_COMPONENT_TYPE will be MSG and OPT_COMPONENT_NAME will be TestFmt. This signals the Rules Daemon to reload the rule set before evaluating this message. The -C MSG is not necessary for this call.

Case 5

```
putdata -i inputfile.txt -a TestApp -m TestFmt -v -s rules -q
InQ -d
```

Uses "rules" session to connect to database. Puts message from inputfile.txt onto "InQ" queue with OPT_APP_GRP and

OPT_MSG_TYPE set. OPT_SHUT_DOWN will be set to TRUE. This signals the Rules Daemon to shut down after evaluating this message.

getdata

Syntax

```
getdata -o <output filename> -q <queue name> [-s <session
name>] [-p] [-v]
```

Description

The NEONet getdata process reads a NEONet queue, retrieving messages one at a time, and writing each message to the output file until the queue is empty.

Operational Assumptions

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

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. The getdata program expects to have a session name of "output" (unless otherwise specified). Using this connection data, the Rules Engine test programs can make a connection to the appropriate database.

Example Configuration File

```
output:MyServerName:MyUserName:MyPassword:MyDB
```

Parameters

Name	Mandatory/ Optional	Description
-o <output filename>	Mandatory	The output file to which each messages are written. The user should have write privilege in the directory in which the getdata program is executed.
-q <queue name>	Mandatory	The NEONet queue name from which the program reads messages.
-s <session name>	Optional	Session name tag from the sqlsvses.cfg file. The default session name is output.
-p (write options)	Optional	Writes the queue options to the output file. The default is to write the queue options to the screen.
-v (verbose)	Optional	The NEONet getdata program logs to the screen if this optional parameter is set. The process defaults to no logging.

Example getdata calls

Case 1

```
getdata -o outputfile.txt -q OutQ -v
```

```
Uses default session "output" to connect to database. Puts all
messages from "OutQ" queue to outputfile.txt. The message
(queue) options will be sent to standard out.
```

Case 2

```
getdata -o outputfile.txt -q OutQ -v -s rules -p
```

```
Uses "rules" session to connect to database. Puts all messages
from "OutQ" queue to outputfile.txt. The message (queue)
options will be sent to outputfile.txt as well.
```

ruletest

The ruletest program can be used to read a message from a file and evaluate the message using the Rules APIs. This test program is useful in that it does not use Messaging and Queuing or Formatter to execute subscriptions.

Syntax

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

Description

The NEONet ruletest program reads a message from a file and evaluates the message using the application group and message type defined on the command line or by answering the prompts provided that parameters were not used. After evaluation, subscriptions are retrieved as they would normally be retrieved and output to the screen, but are not executed using Messaging and Queuing and Formatter.

Operational Assumptions

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

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.

The session name in the sqlsvses.cfg file is used by ruletest to locate the appropriate line from which to retrieve connection data. For rules and formatter data, the ruletest program expects to have a session name of rules (unless otherwise specified).

Example

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

Parameters

Name	Mandatory/ Optional	Description
-i <input filename>	Mandatory	The input file from which ruletest reads. 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 NEONet ruletest program requires this parameter in order to evaluate rules.
-m <message type>	Mandatory	The NEONet ruletest program requires this parameter in order to evaluate rules.
-s <session name>	Optional	Session name tag from the sqlsvses.cfg file. The default session name is rules.
-v (verbose)	Optional	The NEONet ruletest program logs to the screen if this optional parameter is set. The process defaults to no logging.
-? (usage)	Optional	The NEONet ruletest program displays all usage parameters.

Ruletest can be executed using two methods:

1. Ruletest evaluates the message using the specified application group and message type if the user enters the parameters listed above at command time.
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 so many evaluations may be done using the same session.

   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 without any command line parameters, it prompts the user for the required information.

Example ruletest calls

Case 1

```
ruletest -i inputfile.txt -a TestApp -m TestFmt -v

Uses default session "rules" to connect to database. Evaluates
message from inputfile.txt against rule set defined by TestApp
and TestFmt. Does not perform any subscription actions, just
lists them.
```

Case 2

```
ruletest -i inputfile.txt -a TestApp -m TestFmt -v -s test

Uses "test" session to connect to database. Evaluates message
from inputfile.txt against rule set defined by TestApp and
TestFmt. Does not perform any subscription actions, just lists
them.
```

Case 3

```
ruletest

Prompts for the required input: session name, input filename,
application group name, message type name.
This will continue to evaluate messages until an empty line is
entered for the required input.
```

NNRSignalShutdown Utility

Syntax

```
NNRSignalShutdown -q <queue name> [-s <session name>] [-v]
```

Description

The NNRSignalShutdown Utility supplies a graceful way to shutdown the Rules daemon. NNRSignalShutdown puts an empty message with the option OPT_SHUT_DOWN set to TRUE on an input queue for the Rules daemon.

Parameters

Name	Mandatory/ Optional	Description
-q <queue name >	Mandatory	The NEONet queue name where the shutdown message is added. This should match an input queue for the ruleng or the name specified with the -R option for the Rules daemon.
-s <session name>	Optional	The session name tag from the sqlsvses.cfg file. The default session name is input.

Name	Mandatory/ Optional	Description
-v (verbose)	Optional	The NEONet NNRSignalShutdown program will log to the screen if this optional parameter is set. The process defaults to no logging.

Example NNRSignalShutDown calls

Case 1

```
putdata -i inputfile.txt -a TestApp -m TestFmt -v -s rules -q
InQ -d
```

Uses "rules" session to connect to database. Puts message from inputfile.txt onto "InQ" queue with OPT_APP_GRP and OPT_MSG_TYPE set. OPT_SHUT_DOWN will be set to TRUE. This signals the Rules Daemon to shut down after evaluating this message.

Case 2

```
NNRSignalShutDown -q InQ -v -s rules
```

Uses rules session to connect to database. Puts message on "InQ" queue with OPT_SHUT_DOWN will set to TRUE. This signals the Rules daemon to shut down.

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 using the MQSeries Integrator Rules Daemon, you must

- Complete a valid MQSeries installation
- Create all MQSeries queues
- Create a parameter file containing configuration information
- Enter and save rules and formats.

Rules and formats are used by the MQSeries Integrator Rules daemon as defined in this section. A utility is provided to allow you the option of encrypting the UserId and Password for the parameter file.

Queues

The MQSeries Integrator Rules daemon uses input and output MQSeries queues. Input queues are specified by name in the parameter file. Multiple input queues can be defined, each with its own set of default values. 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 use an MQSI header with these two options set:

```
OPT_APP_GRP
OPT_MSG_TYPE
```

OPT_APP_GRP assigns the message to an application group and must match the application group name in the NEONRules database. 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 and defaults are not set, the evaluation will not occur and failure processing occurs.

When a MQSI header is not used, or these options are not set, the MQSeries Integrator Rules daemon assigns defaults based on parameter settings from the parameter file name specified on the command line at startup. The DefaultMsgType parameter can be set to use the message's MQMD field value or any format defined in the NEONFormatter database.

The MQSIruleng.mpf is a parameter file sample provided to guide you in creating your own parameter file. The parameter file contains required and optional parameters, some of which can be tuned to customize control and performance to your environment. See *MQSIruleng* on page 198.

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). MQSeries Integrator 1.0 supports the Reload operation only for reloading an entire application group. MQSeries Integrator 1.1 allows individual components of an application group to be reloaded. Any rules added or changed after the MQSeries Integrator Rules daemon startup are not used until the Reload message is processed.

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 NEONRules 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.

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.

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.

Using the MQSeries Integrator Rules Daemon

MQSIruleng

Syntax

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

Parameters

The parameters described in the following table are used to configure the MQSIruleng via a parameter file. The parameters are tunable, meaning that their values can be adjusted to customize control and performance to your environment. The parameter file allows you to enable and disable optional features and to set values of some required features.

The parameters are divided into five areas: Operations, Logging, Queues, Queue Handle Cache, and Rules Database Connection. Within the parameter file, the parameters are presented in the same groupings. The group heading must be displayed in the parameter file using square brackets. To view a tunable parameter file example, see *Example* on page 205.

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*).

* The default values for these parameters are recommended and are designed to be more than adequate in most environments.

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
Credentials Enabled	Optional	Value determines whether messages are put with alternate user authority. The default is off (value is zero (0)) indicating that messages are put using daemon authority. When turned on (value is one (1)), messages are put using the publisher's authority.
QueueManager Name	Optional	Name of the local MQSeries Queue Manager. If not specified, the MQSeries default queue manager is used.
MaxBackoutCount	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. When multiple input queues are specified, use "first,second,third..." where first, second, and third are the names of the queues. No white spaces are used within the queue list.
ServiceScheme	Optional	Used with Input Queue Name to specify the queue service scheme across specified input queues. "RoundRobin" (the default) processes the first message from each queue in strict rotation. "Drain" processes all messages in the first queue before processing from the next queue.

Name	Mandatory/ Optional	Description
ServiceInterval	Optional	The integer number of seconds the daemon sleeps when any input queue is empty before continuing the progression of the ServiceScheme. The default is one (1) second.
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 to be used for messages without a MQSI header. A DefaultAppGroup value must be supplied by the user; no default value. Mapping of multiple default values for multiple input queues is from left to right. If insufficient default values are specified, the value list is reprocessed from the beginning until sufficient values are assigned. Excess default values are ignored

Name	Mandatory/ Optional	Description
DefaultMsgType	Mandatory	Indicates the message type value to be used for messages without a MQSI header. A DefaultMsgType value must be supplied by the user; no default value. The special values: "$MQMD.Format", "$MQMD.PutApplName", or "$MQMD.ApplIdentityData" can be specified. These values cause the daemon to use the MQSeries Message Descriptor (MQMD) values indicated at runtime. Mapping of multiple default values for multiple input queues is from left to right. If insufficient default values are specified, the value list is reprocessed from the beginning until sufficient values are assigned. Excess default values are ignored

Queue Handle Cache

Name	Mandatory/ Optional	Description
MaxHandles	Optional	Integer value specifying the maximum number of entries allowed in the queue handle cache. When this number of handles has been stored in the cache, the next attempt to insert a queue handle into the cache initiates a cache purge operation.
PurgeInterval	Optional	Integer number of seconds to wait before attempting a cache purge operation. This value is used only when the MQSI rules daemon is sleeping. If the input queues remain full, MaxHandles dictates when a cache purge occurs.

The Queue Handle Cache parameters described above are not required for the MQSI Rules Daemon to run. However, use of these tunable parameters is recommended to optimize performance. When MaxHandles and PurgeInterval are not specified, the daemon default is to disable its cache. The cache is purged using a Least Recently Used (LRU) algorithm.

If the cache is full, but no entries match the purge selection criteria, a random entry is selected for removal from the cache.

Rules Database Connection

Name	Mandatory/ Optional	Description
Server Name	Mandatory	The name of the server that you want to connect to. For Oracle, this is optional. For DB2, enter the database name here and leave Database Instance as "???".
User ID	Mandatory	Your User ID. Can be encrypted using the MQSIencrypt utility.
Password	Mandatory	Your password. Can be encrypted using the MQSIencrypt utility.
Database Instance	Mandatory	The name of the database that you want to connect to. Leave as "???" for DB2 or Oracle.
Database Type	Mandatory	Type the number of the database type: SYBASE with CTLIB = 1 SYBASE with DBLIB = 2 ORACLE 7.x = 3 MSSQL = 4 DB2 = 5 ODBC = 6

Remarks

MQSeries Integrator uses parameter files of the following structure:

Note:
You can not have trailing whitespace after a group identifier because the file fails to parse correctly. You must have a return immediately following the closing bracket of the group identifier. An example of a group identifier as shown below is: `[Group1]`

```
[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
              .
              .
              .
```

Note:
This spaces before and after the "=" are required.

Example

```
############################################################
#
# This is the tunable parameters file for MQSIruleng.
#
# Comments must have a number sign(#) in the first
# column. Trailing comments are forbidden.
#
# 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
```

```
# Alternate User Authority flag
CredentialsEnabled       = 0

## MQSeries queue manager name...defaults to default queue
## manager
#QueueManagerName        = ???

# number of replay/retry limit before failed 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        = ???

[Queue Handle Cache]
## parameters used to control the output queue handle cache...
# MaxHandles             = 25
# PurgeInterval          = 30

[Logging]
# Log file control..."LogFileName" is the file specification
for
# the log, and valid "LogLevel" settings 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]
#
# Rules and Formatter database connection information
# (mandatory)
```

```
# Exceptions/notes:
#   - leave "DatabaseInstance" as "???" (Oracle and DB2 only)
#   - enter the database name as the value of "ServerName" (DB2
#     only)
#
ServerName        = ???
UserId            = ???
Password          = ???
DatabaseInstance  = ???
#
# DatabaseType is a numeric with these values:
#   SYBASE (CTLIB bindings)  = 1
#   SYBASE (DBLIB bindings)  = 2
#   MSSQL                    = 4
#   DB2                      = 5
#   ODBC                     = 6
#   ORACLE (version 7.x)     = 8
#   ORACLE (version 8.x)     = 9
DatabaseType    = ???
#
# end of file!
#
```

Encrypting the Parameter File

After the parameter file is created, the MQSIencrypt utility can be used to encrypt the UserId and Password keys of the Rules Database Connection parameter group. The utility creates a copy of the parameters file with the UserId and Password encrypted. Upon verification that the encrypted parameter file is operational, the original plaintext file should be deleted. Encryption does not affect the procedure used to start the daemon. The parameters file can be edited after encryption to modify all other parameter key / value pairs, provided you do not alter the encrypted values.

Running the Rules Daemon

MQSIruleng can be started and stopped in UNIX or as an NT Service.

Running ruleng on UNIX

To start the MQSI Rules daemon, either pass in the commands on the command line or follow the prompt for the parameters.

Running ruleng as an NT Service

You can run MQSIruleng as a Service under Windows NT 4.0.

The benefits of running ruleng as a service include the ability to:

- Start, stop, and pause the service from the service GUI provided by NT, or from the command line.
- Run multiple services at one time.
- Schedule a service to start automatically when a machine is unattended.

Naming Your Services

Before installing your services, you must determine a unique name for each service to be installed. The service name points any service commands to the appropriate messaging and queuing engine or the appropriate rules engine.

Consider the following:

- Uniquely name each entity in relation its function or component rather than relying on case sensitivity to distinguish them. To prevent potential database conflicts, use case sensitivity as a readability factor but not as the sole differentiation between names. While TestCase, TESTCASE, and testcase are each considered as unique names in a case-sensitive database, they are duplicates within a database that is not case sensitive.
- Choose short names that are meaningful to your systems and organization.
- Capitalize each word in a name to make it more readable and to serve as a consistency standard (such as *TestImport*.)

Notes:
The service names and corresponding executables are stored in your system's registry. It is not recommended that the registry executables be changed or modified.

Installing NT Services

Services are named and installed from the command line and must be installed before you can run them. The following install syntax handles the naming of your service and its installation.

```
MQSIruleng -install <your service name> <service options> -p
<parameterFileName>
```

Detailed information about available Service Options is found on page 211. Upon installation, whenever the service is started, it will reference and run using the parameters set in the parameter file.

When the installation is complete, a confirmation message appears. If installation is unsuccessful, an error message appears.

To verify that the service is installed and is available, open the NT services window, as described in the following section. To further check the status of the service and display the default parameters / options, use the following command:

```
MQSIruleng -info <your service name>
```

Example:

To install a service named RulesWest for MQSIruleng without service options, type:

```
MQSIruleng -install RulesWest -p MQSIruleng.mpf
```

Managing Services from the NT GUI

From the Windows NT desktop, select Start→Settings→Control Panel. Double-click Services in the Control Panel window.

The NT Services window appears. Installed Services are listed in the window. For each Service, its current status and startup method is listed.

Running a Service

1. From the NT Services window, highlight the Service you want to start. Click Start. The Status changes to Started.

2. To Stop or Pause a Service, highlight the Service and click Stop or Pause. The Status will reflect the change.

Scheduling Automatic Startup for a Service

Services can be scheduled to start automatically according to parameters you set. Highlight the service you want to schedule and click Startup.

1. Choose the Startup Type.

   Automatic: Service runs automatically when the system starts. The Service will start only if the computer has 12MB or more of random access memory (RAM).

   Manual: Service runs only when started by a user or a dependent service. Service remains running until it is stopped, even if the user that started the service has logged off the system.

   Disabled: Service is disabled and will not start.

2. Identify the Log On As parameters.

   System Account: The Service logs on to a system account versus a user account. Most Services log on to a system account.

   This Account: The Service logs on to a specific user account with corresponding password. Click the browse button to specify a user account, and then type the password for the user account in both the Password and Confirm Password boxes.

   To provide a user interface on the desktop that can be used by whoever is logged in when the service is started, select the Allow Service to Interact with Desktop check box.

3. Click OK.

Managing Services from the Command Line

The command line can provide more detailed descriptions of errors when they occur.

Notes:
The GUI does not automatically refresh after a command line operation is performed affecting a service. Close and reopen the GUI to view the change to the service.

Syntax

```
MQSIruleng <service option> <service name> <standard option
(optional, as needed)> -p <parameterFileName>
```

Service Options

Service Option	Description
`-install`	Installs the service with the <standard option> used as startup defaults.
`-remove or -uninstall`	Removes the service.
`-start`	Starts the service. The <standard options> override default startup parameters.
`-stop`	Stops the service.
`-pause`	Pauses the service if it is running.
`-continue`	Continues the service if it is paused.
`-info`	Prints service status and current default parameters.
`-parameters`	Installs <standard options> as the default startup parameters the next time the service is started. To change parameters for one time only, use the GUI.
`-? or -help`	Displays service usages and printing of usage text.

Examples

Starting a Service in ruleng

To start a service named RulesWest in MQSIruleng with no options:

```
MQSIruleng -start RulesWest
```

By starting the service from the command line versus through the GUI, the default startup options will be overridden by no options in this command. This occurs for this service start only because the default startup parameters are not permanently changed by the command line start.

To Uninstall the Service from ruleng

To uninstall RulesWest as a service for MQSIruleng, type:

```
MQSIruleng -uninstall RulesWest
```

RulesWest will no longer be available as a service and will not appear on the NT Services window.

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. Modify the line:

   ```
   inputFileName = null.dat
   ```

3. Create a null file named "null.dat"
4. 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.

Notes:

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, including all connections to queues in the output queue handle cache, are not shutdown using *Ctrl+c*.

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.

Notes:
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 the parameter file. 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 reported to the Log File only if a log file is specified. If no log file is specified, errors appear on the screen of the process which started the MQSI Rules Daemon.

Caching Output Queue Handles

Queue handle caching improves the performance of put queue actions requested by rule subscriptions. Tunable parameters define the cache.

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 do not contain a data segment (header and options only) 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. Modify the line:

   ```
   inputFileName = null.dat
   ```

5. Exit the MQSIputdata.mpf parameter file.
6. Create a null file named "null.dat"
7. 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
```

MQSeries Integrator Rules Daemon Error Messages

Key to Message Codes and Severity

Message Type	Error Code Range	Severity Code
Information	10000 - 10099	0
Warning	10100 - 10199	1
Error	10200 - 10299	2
Fatal	10300 - 10399	3

Informational Messages

Code	Message	Message Severity	Response
10000	Received the input message with input queue: <insert character string> 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.

Code	Message	Message Severity	Response
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.
10011	A putqueue action has completed.	0	None. This is an information message.

Code	Message	Message Severity	Response
10012	Shutdown request detected.	0	None. This is an information message.
10013	Reload of a rule set completed successfully. Application Group: <insert character string> Message Type: <insert character string>	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.
10017	Reload of rules component completed successfully. Application group: <insert character string> Message Type: <insert character string> Component Name: <insert character string>	0	None. This is an information message.
10018	Component not found or rule set not currently in cache. Reload request ignored. Application group: <insert character string> Message Type: <insert character string> Component Name: <insert character string>	0	None. This is an information message.
10019	Reserved for use by IBM.	0	NA

Code	Message	Message Severity	Response
10020	Getqueue action normal completion. No more messages available.	0	None. This is an information message.
10021	Getqueue action complete. Actual number of messages read: <insert number> Number of messages specified in parameters file: <insert number>	0	None. This is an information message.
10022	Encryption operation completed successfully.	0	None. This is an information message.
10023	Statistics at the end of the message put operation: Number of messages put: <insert number> Number of commits done: <insert number>	0	None. This is an information message.
10024	Start information related to the put operation: Queue name: <insert character string> Number of messages to be put: <insert number or "ALL"> Count of each messages to be put: <insert number> Commit size:<insert number>	0	None. This is an information message.
10025	Commit operation succeeded. Number of messages put/ committed: <insert number> Target queue name: <insert character string>	0	None. This is an information message.

Code	Message	Message Severity	Response
10026	Statistics at the end of message get operation. Number of messages got: <insert number> Number of commits done: <insert number>	0	None. This is an information message.

Warning Messages

Code	Messages	Severity	Response
10100	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.
10101	Message being processed exceeds current buffer size. Message size: <insert number> Current buffer size: <insert number>	1	Increase the MaxBufferSize.
10102	Reserved for use by IBM.	1	NA
10103	The queue <insert character string> is full. No more messages can be put.	1	Check to make sure messages are being read from the queue so that more messages can be put. Consider increasing MaxDepth (see MQSeries documentation for information on setting the MaxDepth queue command).

Code	Messages	Severity	Response
10104	The specified commit size <insert number> exceeds the maximum permissible commit size <insert number>. Setting the commitSize to <insert number>.	1	Reduce commitSize to a value less that the system constraint. The commitSize is changed to the system constraint value as indicated in the error message.
10105	The current record size <insert number> exceeds the maximum permissible record size <insert number>. Cannot put the message into the queue.	1	Set maxUserDataLength in MQSIputdata.mpf to a value at least equal to the message size of the largest message expected.
10106	Message segmentation not allowed for messages having header format <insert character string>. Disabling the message segmentation feature.	1	Verify that the specified format is supported for message segmentation.

Error Messages

Code	Message	Severity	Description
10200	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.

Code	Message	Severity	Description
10201	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.
10202	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.
10203	Output type missing for subscription <insert number>.	2	Verify that the output format is specified for this subscription.
10204	Input type missing for subscription <insert number>.	2	Verify that the input format is specified for this subscription.
10205	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.

Code	Message	Severity	Description
10206	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.
10207	Target queue not set for putqueue action.	2	Verify that the putqueue action in your Rules subscription has a target queue defined.
10208	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. This message applies to releases of MQSI prior to 1.1. With MQSI 1.1 messages are not rejected.

Code	Message	Severity	Description
10209	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.
10210	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.
10211	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.
10212	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.

Code	Message	Severity	Description
10213	Putqueue action failed: Subscription Action: <insert character string> Subscription ID: <insert number>	2	Check the log for additional details about this error.
10214	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.
10215	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.
10216	The Publish operation failed.	2	Check the log for additional information about this error.

Code	Message	Severity	Description
10217	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.
10218	The input message conversion failed. The message will be sent to the failure queue.	2	Check the Log file for details.
10219	Reload of rule set failed. Application group: <insert character string> Message Type: <insert character string> Rules Error #: <insert number> Rules Error Message: <insert character string>	2	Check the Log file for details.
10220	Subscription name missing from reload message.	2	Check the reload message.
10221	Invalid reload component specified in reload message.	2	Check the Log file for details.

Code	Message	Severity	Description
10222	Reload of rule set failed. Application group: <insert character string> Message Type: <insert character string> Rules Error #: <insert number> Rules Error Message: <insert character string>	2	Check the Log file for details.
10223	Cannot obtain output queue handle from cache for queue.	2	Internal error. Refer to the MQSI log file for details. Contact product support.
10224	Encryption algorithm cannot process specified string.	2	Re-encrypt a plaintext .mpf to determine if error is corrected. Refer to the MQSI log file for details. Contact product support.
10225	Failed during encryption operation. File Name: <insert character string> Line Number: <insert number>	2	Contact product support.
10226	The record separator string is not specified in the tunable parameter file. It is mandatory for variable length record.	2	Verify that recordSeparator is set as an ascii string in MQSIputdata.mpf.

Fatal Error Messages

Code	Message	Severity	Description
10300	Bad condition code detected.	3	Internal error. Contact product support.
10301	Unrecognized output type in failure message. Message: <insert character string>	3	Contact product support.
10302	A Null Log File handle was detected.	3	Internal error. Contact product support.
10303	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.
10304	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.
10305	Failure queue not specified.	3	Verify that a failure queue is defined by the MQSeries Integrator Rules daemon configuration file.
10306	No input queues were specified.	3	Verify that an input queue is defined by the MQSeries Integrator Rules daemon configuration file.

Code	Message	Severity	Description
10307	No Hit queue not specified.	3	Verify that a No Hit queue is defined by the MQSeries Integrator Rules daemon configuration file.
10308	Unexpected argument: <insert character string>	3	Verify the syntax of the command used to run the MQSeries Integrator Rules daemon.
10309	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.
10310	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.
10311	Failed to commit publish operation. 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.

Code	Message	Severity	Description
10312	Failed to rollback publish operation. 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.
10313	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.
10314	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. Check with your System Administrator to verify that the operating system is correctly configured.

Code	Message	Severity	Description
10315	Message being processed exceeded maximum allowed size. Message size: <insert number> Required buffer 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.
10316	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.
10317	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.
10318	Could not create formatter object.	3	Check the log for additional information. Refer to the Formatter documentation for additional information on causes of this error.
10319	Could not create rules object.	3	Refer to the rules documentation for additional information on causes of this error.

Code	Message	Severity	Description
10320	Could not access parameter file. Parameter file name: <insert character string>	3	Verify that the MQSeries Integrator Rules daemon parameter file exists.
10321	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.
10322	Failed to open rules session. Error #: <insert number> Error Message: <insert character string>	3	Diagnose problem based on Error # and Error Message.
10323	Usage: <insert character string> -p filename	3	Check the command line invocation of the MQSeries Integrator Rules daemon.
10324	Default application group and/or message type not specified.	3	Define the DefaultAppGroup and DefaultMsgType parameters in the Rules daemon configuration file.

Code	Message	Severity	Description
10325	Failed to complete inquiry. Target Object Name: <insert character string> Inquiry Code: <insert number> MQSeries Conditions Code: <insert number> MQSeries Reason Code: <insert number>	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.
10326	Cannot initialize default parameters.	3	Check the MQSI log file and / or screen for additional messages with more detail.Correct described errors in the parameters file. Restart the daemon.
10327	Unknown internal error encountered. File Name: <insert character string> Line Number: <insert number>	3	Contact product support.
10328	Error disconnecting from queue manager. Queue Manager 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.

Code	Message	Severity	Description
10329	Insufficient space for options. File Name: <insert character string> Line Number: <insert number>	3	Set "MaxUserDataLength" to a larger value in MQSIputdata.mpf and retry the message put.
10330	Error closing queue. Queue Name: <insert character string> Completion Code: <insert character string> 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.
10331	No output queues specified in the parameters file.	3	Specify the output target queue name in MQSIputdata.mpf.
10332	The parameter file has an invalid or unsupported value. Parameter File Name: <insert character string> Parameter Value: <insert character string>	3	Correct the value indicated and restart the MQSI Rules Daemon.
10333	Error opening a queue manager for inquire operation. Queue manager 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 retry the MQSIputdata operation.

Code	Message	Severity	Description
10334	Error closing a queue manager after inquire operation. Queue manager 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 retry the MQSIputdata operation.
10335	Error during inquire of an MQSeries object. Object type: <insert character string> Object 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 retry the MQSIputdata operation.
10336	No input file name specified.	3	Verify that inputFileName is set in MQSIputdata.mpf.
10337	No output file name specified.	3	Verify that outputFileName is set in MQSIgetdata.mpf. outputFileName is required when outputToFile is set to a value of one (1).
10338	Failed to open the specified file <insert character string> for input.	3	Verify that inputFileName is set in MQSIgetdata.mpf to a valid file containing messages.

Code	Message	Severity	Description
10339	Failed to open the specified file <insert character string> for output.	3	Verify that outputFileName is set in MQSIputdata.mpf to a valid file.
10340	Rules object creation error. Error number: <insert number> Error message: <insert character string>	3	Based on the error message text, correct the error and retry, or contact product support.
10341	Formatter object creation error. Error number: <insert number> Error message: <insert character string>	3	Based on the error message text, correct the error and retry, or contact product support.

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 Oracle SQLPlus, Sybase ISQL, or Informix DB Access installed to run the Consistency Checker using UNIX/Korn Shell commands. Microsoft SQL Server is also a supported DBMS type.

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

Rules

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

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

Notes:

- The database name is not needed for Oracle.
- 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 NEONFormatter, type the following UNIX/ Korn Shell command:

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

Reports

The Consistency Checker for NEONFormatter and NEONRules 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 NEONRules 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 NEONRules where data was forcibly corrupted.

Message types referring to nonexistent application group:

Message Type	Message Id	Application Id
CCFlat 1	50	150

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 Report: Rules

The NEONRules Consistency Checker report provides the following information:

Consistency Checker Report	Explanation
Arguments that refer to nonexistent Boolean operators	Boolean operator does not exist for the argument. This will cause load failures.
Arguments that refer to nonexistent operations	The argument's operation does not exist. This may cause load or evaluation failure.
Arguments that refer to nonexistent operators	The operator does not exist for the argument. This will cause evaluation failure.
Arguments that refer to nonexistent rules	The rule does not exist for the argument. This may cause load failures.
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.
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.
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.
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 refer to nonexistent rules	The rule does not exist for the argument. This may cause load failures.

Consistency Checker Report	Explanation
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.)
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.)
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.
Hierarchy definitions that are not complete for Rule / Subscription Permissions	The hierarchy definitions must be complete during the installation of NEONRules with Permissions.
Message types in Rules that do not match a Format in Formatter	The message type does not correspond to any input format in the NEONFormatter. The format may have been deleted in Formatter. Do not use Rules in this Message Type.

Consistency Checker Report	Explanation
Message types in Rules that do not match a Format in Formatter	The message type does not correspond to any input format in the NEONFormatter. The format may have been deleted in Formatter. Do not use Rules in this Message Type.
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.
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.
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.
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.'
Permission Checks	
Permissions granted to nonexistent Item	Rule / Subscription permissions must refer to a valid item name.

Consistency Checker Report	Explanation
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 that are not complete	Rule / Subscription permissions must include Node, Application Group, Message Type, and Rule / Subscription Name to be complete.
Permissions that do not exist in the hierarchy	Rule / Subscription permissions must refer to valid hierarchy information.
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 Nodes	Rule / Subscription permissions must refer to the current node.
Permissions that refer to nonexistent Rules	Rule permissions must refer to a valid rule name.
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.
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.

Consistency Checker Report	Explanation
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.
Rules with multiple owners	Rules can only have one owner. If 'PUBLIC' is the rule owner, every user has de facto ownership.
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.
Rules with no Owners	Each rule must have a single owner. A rule with 'PUBLIC' as its owner is basically owned by everyone.
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.
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.
Subscription with multiple owners	Subscription can only have one owner. If 'PUBLIC' is the rule owner, every user has de facto ownership.

Consistency Checker Report	Explanation
Subscription with no Owners	Each Subscription must have a single owner. A subscription with 'PUBLIC' as its owner is basically owned by everyone.
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.
Subscriptions that refer to nonexistent rules	The rule does not exist for the subscription. This may cause load failure.
Subscriptions with No Actions	All subscriptions must have at least one action. This report displays subscriptions with no actions. This may cause evaluation failure.
Unique Sequence Generator invalid for Permission Users	The system cannot add additional users for permissions because it cannot generate a unique identifier.
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.
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).

Note:
When running the MQSeries Integrator Rules daemon, subscriptions for rules that hit should end with a Put Message action to route the message. This might not be needed if users provide their own daemon and generic actions.

Consistency Checker Report: Formatter

The NEONFormatter Consistency Checker report provides the following information:

Consistency Checker Report	Explanations
Case operations that refer to nonexistent case choices	Choose valid choices for case operations.
Case operations 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.
Collection operation components that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.
Collection operations that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.
Collection type output operations that have no collection components	Choose at least one component operation to insert into a collection.
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 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 repeat fields	Choose valid fields for "Field contains repeat count" repeat termination.

Consistency Checker Report	Explanations
Compound formats that have no component formats	Insert at least one component format into compound format.
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.
Exit operations that refer to nonexistent output operations	Extraneous data in the database. 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 format delimiters	Choose valid delimiters for flat formats.
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 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.
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.
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.

Consistency Checker Report	Explanations
Input compound format components that refer to nonexistent repeat terminations	Choose valid repeat termination types for component formats.
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.
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 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 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 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 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.
Input controls that have invalid default date format strings	Date data type refers to a time format string that is not the legitimate default.

Consistency Checker Report	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 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 data delimiters	Choose valid literals for data delimiters of 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 data types	Choose valid data types for data portion of input parse control.
Input controls that refer to nonexistent input control types	Choose valid types for input parse controls.
Input controls that refer to nonexistent length data types	Choose valid data types for length portion of input parse control.
Input controls that refer to nonexistent length delimiters	Choose valid literals for length delimiters of input parse controls.
Input controls that refer to nonexistent length locations	Choose valid length locations 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 data types	Choose valid data types for tag portion of input parse control.
Input controls that refer to nonexistent tag delimiters	Choose valid literals for tag delimiters of 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.

Consistency Checker Report	Explanations
Input controls that refer to nonexistent tag termination types	Choose valid tag termination types for input parse controls.
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.
Input format fields that refer to nonexistent input controls	Choose valid input parse controls for the fields.
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.
Justify operations that refer to nonexistent justify choices	Choose valid choices for justify operations.
Justify operations that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.
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 components that refer to nonexistent math expression operations	Extraneous data in the database. Database integrity may be compromised.
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.

Consistency Checker Report	Explanations
Output compound format components that refer to nonexistent repeat terminations	Choose valid repeat termination types for component formats.
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 date format strings	Date data type refers to a date 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 refer to nonexistent calculation operations	Choose valid calculation operations for output format controls.
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 controls that refer to nonexistent data types	Choose valid data types for data portion of 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 length data types	Choose valid data types for length portion of 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 tag data types	Choose valid data types for tag portion of output format controls.

Consistency Checker Report	Explanations
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 format fields that refer to nonexistent access modes	Choose valid access modes for fields in flat output formats.
Output format fields that refer to nonexistent fields	Choose valid fields to insert into flat output formats.
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 input fields	Choose valid mapped input fields to insert into flat output formats.
Output format fields that refer to nonexistent output controls	Choose valid output format controls for the fields.
Output operations that refer to nonexistent case 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.
Output operations that refer to nonexistent default 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 length 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 operation types	Database integrity is compromised. Delete operation and re-enter it.

Consistency Checker Report	Explanations
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 substitute 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.
Output operations that refer to nonexistent trim 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.
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 prefix / suffix choice	Choose valid choice for prefix / suffix operation.
Prefix / suffix operations that refer to nonexistent prefixes or suffixes	Choose valid literals for prefixes or suffixes.
Substitute operations that refer to nonexistent input values	Choose valid literals for substitute input value.
Substitute operations that refer to nonexistent output data types	Choose valid data types for substitute output value.
Substitute operations that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.
Substitute operations that refer to nonexistent output values	Choose valid literals for substitute output value.
Substring operations that have invalid substring parameters	Choose a substring start position >= 0 and a substring length > 0.

Consistency Checker Report	Explanations
Substring operations that refer to nonexistent output operations	Extraneous data in the database. Database integrity may be compromised.
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.
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.
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.
User defined data types with invalid data type identifiers	Extraneous data in the database. Database integrity may be compromised.

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.

Data Type Field Value	Description
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).

Data Type Field Value	Description
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.
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.

Data Type Field Value	Description
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.

Data Type Field Value	Description
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.
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

G

I

L

M

N

S

T

U

Sending your comments to IBM
MQSeries Integrator
System Management Guide
SC34-5505-01

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 mail, use the Readers' Comment Form
- 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

Readers' Comments
MQSeries Integrator
System Management Guide
SC34-5505-01

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

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.

Be sure to print your name and address below if you would like a reply.

Name Address

Company or organization

Telephone Email

You can send your comments POST FREE on this form from any one of these countries:

Australia	Finland	Iceland	Netherlands	Singapore	United States
Belgium	France	Israel	New Zealand	Spain	of America
Bermuda	Germany	Italy	Norway	Sweden	
Cyprus	Greece	Luxembourg	Portugal	Switzerland	
Denmark	Hong Kong	Monaco	Republic of Ireland	United Arab Emirates	

If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

1 Cut along this line

2 Fold along this line

By air mail
Par avion

IBRS/CCRI NUMBER: PHQ - D/1348/SO

NE PAS AFFRANCHIR
NO STAMP REQUIRED

IBM

REPONSE PAYEE
GRANDE-BRETAGNE

IBM United Kingdom Laboratories
Information Development Department (MP 095)
Hursley Park
WINCHESTER, Hants
SO21 2ZZ United Kingdom

3 Fold along this line

From: Name _____
Company or Organization _____
Address _____
_____
EMAIL _____
Telephone _____

1 Cut along this line

4 Fasten here with adhesive tape

Printed in U.S.A

SC34-5505-01